Personalized TV with Recommendations

Integrating Social Networks

MICHEL SHAHAN

KTH Information and
Communication Technology

Master of Science Thesis
Stockholm, Sweden 2008

COS/CCS 2008-25

Personalized TV with Recommendations

Integrating Social Networks

Michel Shahan
shahan@kth.se

Master's thesis

27 Oktober 2008

Examiner and Academic Supervisor
Prof. Gerald Q. Maguire Jr.

Ericsson Industrial Supervisor
Mattias Lidström

Preface

This thesis is part of the requirements for the examination for a Master Science in Engineering at Royal Institute of Technology (KTH). The project was performed at Ericsson AB Research, Service Layer Technology (SLT) during spring-summer term of 2008. The examiner and academic advisor of the thesis at KTH was Professor Gerald Q. Maguire Jr. at Department of Communication System. The supervisor at Ericsson AB was Mattias Lidström.

Abstract

This master's thesis concerns how to recommend multimedia content which a user might view – with some media player. It describes how a computer application (called a recommendation engine) can generate better recommendations for users based on using information available from social networks and the media selections made by others.

This thesis gives an introduction to the area of "recommendations", recommendation engines, and social networks. An overview of existing recommendation techniques suggests potential solutions to the problem of what recommendations to make to a given user. The thesis presents how social networks can be used to further enhance the users' experience and describes the work that has been done to realize this recommendation system. An evaluation of the implemented solution is given. The thesis concludes with a summary of how recommendation engines and social network technologies can be used and suggests some future work.

This thesis is of current interest since there is a tremendous quantity of content which is being offered in stores and via services on the web. A recommendation system makes it easier for users to find content which they find appropriate. Since social network communities are growing rapidly there has been an interest to use this information to get recommendations from friends. The results from the evaluation of the prototype recommendation system show how social networks which utilize trust systems might affect the recommendation which is given.

Sammanfattning

Det här examensarbetet behandlar hur man kan rekommendera multimedia som en användare kan tänkas titta på. Den beskriver hur en dataapplikation (kallad rekommendationsmotor) kan generera bättre rekommendationer åt en användare genom att använda information från sociala nätverk, och andra användares tidigare val av media.

Examensarbetet ska ge en introduktion in om ämnet ”rekommendationer”, rekommendationsmotorer och sociala nätverk. En överblick av existerande rekommendationstekniker ger potentiella lösningar till problem för rekommendationer till en användare. Examensarbetet ska även presentera hur sociala nätverk vidare kan förbättra användarupplevelsen och beskriva arbeten som har gjorts för att realisera. En evaluering av det implementerade systemet kommer att ges. En slutsats om hur rekommendations motorer och sociala nätverk kan användas och hur man kan fortsätta på arbetet kommer avsluta rapporten.

Examensarbetet är intressant eftersom det finns rikligt av tjänster och produkter på nätet som utnyttjar rekommendationsmotorer. Rekommendationssystem ser till att användare enklare kan hitta information som anses lämplig för användaren. Eftersom även sociala nätverk är en växande trend finns det intresse att använda den här informationen till att ge rekommendationer från vänner. Resultatet från evalueringen av det utvecklade systemet kommer att visa en intressant slutsats om hur sociala nätverk som utnyttjar ”trust” system kan påverka rekommendationen.

Acknowledgement

I would expressing my gratitude towards my KTH supervisor, Professor Gerald Q. "Chip" Maguire Jr., for his help and advice during the thesis, through correction of the report, face to face meetings, and fast e-mail replies.

I would also like to express my gratitude towards Mattias Lidström A and Ericsson AB for offering the opportunity to work on such an interesting Master's thesis project. I appreciate the support that he has given during this time. I would also like to thank the Personal TV project members at Ericsson AB for helping me to evaluate the prototype.

Table of content

1 Introduction

1.1 Background

As the range of content available via different services increases it becomes harder for the user to find what they are looking for and to even find what they consider to be "interesting". The vast amount of content that is available is likely to have some subset which matches the personal interests of a *specific* user. However, a user is unlikely to find this content based upon a random search, thus there is a need for recommendations of content relevant to *this* specific user.

Several recommendation engines have been developed and deployed for different commercial and non-commercial needs. One well known recommendation engine is that utilized by Amazon [41]. In this system the user's shopping behavior is monitored by storing information about past purchases in a database. Because many users' behaviors are stored, the engine can suggest to the user products that he or she is likely to be interested in purchasing.

Recommendation engines have often been developed to infer the user's purchasing habits and estimate which content that a user is likely to be interested in. Different algorithms are used to maximize the accuracy of the recommendation. However, it is increasingly common to look to additional information sources that can be taken into consideration when making a recommendation. By looking at users' habits one can determine what music or movies each user likes, for example by monitoring their web browsing, media player, or phone calls. Knowing this; a recommendation engine can adapt its recommendations to the user by suggesting only the music, films, books, etc. that the user is most likely to be interested in, according to the patterns that were extracted by data mining the usage data of other users along with this user's previous behavior.

1.2 Thesis overview

This master's thesis will focus on how social networks can be used to provide a user with a better user experience in the specific area of *recommended* TV content. The approach adopted in this thesis is based on the assumption that users with relations in real life are likely to want to share their TV watching experience with others, thus a social network based TV recommender should exploit this social correlation (or perhaps the desire for social correlation). The type of the relationship between users can affect which types of recommendations are given. As an example, friends tend to share interests, therefore there is a high probability that content enjoyed by a user is also enjoyed by their friends. This can be specific to a given content type, such as movies, thus a shared interested among a group of friends may lead to one or more recommendations for this type of content and one or more recommendations for other types of content.

The proposed system needs to gather information about the user's relationships with other users. This could be done using one or more resources. This data mining of relationships leads to a number of privacy issues. Next the usage patterns of the content must be gathered which raises even more privacy issues.

As social networks play a major role in this thesis, the thesis examines how to integrate information obtained from social networks with current recommendation engines. This involves deciding what type of recommendation technique is most appropriate and what parts of the engine to modify in order to be able to integrate this new source of information.

Another question that has to be answer by this thesis is what the users explicitly have to do to get a recommendation that is relevant to them (*if any*) and how the users will interact with the system.

2 Overall architecture

The overall architecture connects the different modules examined in this thesis. It also determines how they will interact with each other. The thesis is divided into three basic modules:

- Social network extraction
- Relationship inferring
- Integration with a recommendation engine

Social network extraction involves gathering data from different sources to be able to create a social network. Once the social network information has been gathered, the next step is to predict the relationship of each user to the other users. This predicted relationship does not mean that these users actually share a close relationship in real life, but only that one user would recommend something that another related user is likely to be interested in. Thus these predicted (inferred) relationships may not (and *need not*) reflect real life relationships.

A directed relation between two users will be indicated by a numerical weight (also referred to as a rank). This weight will be used to strengthen the influence of one user (the ranked user) in the recommendation to the other user (the active user). Higher values will increase the probability that the items which have been rated by the ranked user will be recommended to the active user. Note that here we refer to the user for whom we are generating a recommendation as the *active user* – since we assume that the operation of making a recommendation is a proactive suggestion being made to this user. While the *ranked user* represents a user whose choices have previously been made and recorded.

The system will recommend content using a collaborative filtering algorithm or collaborative filtering combined with the information extracted from the user's social networks. The user should be able to ignore the recommendation. The fact that a user chose to ignore a recommendation can be used as feedback for machine learning (for example to adjust the weights used in making this recommendation, changing the inferred relationship, etc.).

2.1 Social Network Extraction

Social network extraction extracts information about relationships between different users. Given a source (a social network) the extraction should be able to extract the desired information and provide it to the rest of the system. For this thesis, the prototype will have limited number of sources. (Details of these sources will be given in Chapter 9.)

2.2 Relationship Inference

Given a graph of a social network the predictor will *infer* relationships. The output of this inference process is a numeric value which gives an indication of the strength to a specific user.

2.3 Recommendation Engine Integration

The integrated system should be able to utilize the inferred relationship values and previously selected content (selected by the ranked user) in order to generate recommendations for the active user. The recommendations will be split up into two parts: recommendations based upon collaborative filtering *alone* and recommendations based upon collaborative filtering *with information from the social network*. In the latter case the recommendation engine should utilize the numeric relationship values, called *relation ranks*, in order to generate recommendations based on friends' earlier selections of content. (Note that here we refer to the ranked users whose numeric relationship values (relation ranks) exceed some threshold are considered to be **friends**. As noted earlier these might or might not actually be person's that the active user would think of as "friends".)

3 Recommendations

Recommendation engines (see Chapter 4) are becoming more important as the amount of content increases, since the sheer quantity of content makes it hard to find the content that is relevant to a user. Recommendation engines are widely used on commercial websites as a suitable recommendation makes it easier for the user to find a product that they are interested in, which leads to a greater probability of the user making a purchase. Recommendations can be generated based on previous purchases, top selling items, or other techniques. The goal of the recommendation engine is to give each user a personalized recommendation based on the user's profile, along with ancillary information. One of the most successful and popular recommendation techniques is collaborative filtering, which will be explained in section 3.1.

Recommendation engines often utilize a database in which they collect data about the users, content, preferences, and other information that might be of use to the recommendation engine. In this thesis we will augment this traditional data by adding information extracted from social networks. (Details of social networks will be given in Chapter 6).

We will begin by studying recommendation techniques, in order to understand which technique(s) can provide a scalable and accurate recommendation. The selected technique(s) will subsequently be used to create a prototype implementation of a recommendation system which can optionally include information extracted from social networks.

3.1 Collaborative filtering

Collaborative filtering is one of the most used recommendation techniques and it has been successfully implemented in many systems with good results, see Chapter 4. It works by collecting a large set of information about the users, items, and the user's preferences for these items. The idea is to base a recommendation on what other users have shown a preference for, thus the name "collaborative filtering".

The cross product of the set of users (U) and the set of items (I) creates a matrix where all ratings from all the users can be represented. All ratings for an item can be found in the vector created by the columns of the matrix and each user's ratings can be found in the vector created by the row of the matrix. Gaps (undefined values) in the matrix indicate that the user has not indicated any opinion for an item; thus the item might be recommended to this user, but with unknown consequences. With the help of the values in this matrix a predicted preference can be calculated for each user for each item. Table 3-1 shows a user-item matrix of M users and N items. The matrix shows that two users, $User_1$ and $User_3$, have rated the same item ($Item_2$). All items which $User_1$ has rated and $User_3$ has not rated will be candidates for recommendation, which in this example is $P_{3,1}$.

Table 3-1: A user-item matrix which shows that $User_1$ is a candidate to recommend items to $User_3$.

		Items			
	Ratings	$Item_1$	$Item_2$	…	$Item_N$
	$User_1$	$R_{1,1}$	$R_{1,2}$	…	
	$User_2$			…	
Users	$User_3$	$P_{3,1}$	$R_{3,2}$	…	
	…			…	
	$User_M$			...	

Similarity between users and items are calculated using **correlations**. This similarity can be split into user based and item based similarity. The only difference between them is that the user based similarity is calculated based upon the correlation between the rows in the matrix and the item based similarity is calculated based upon the correlation between the columns in the matrix. There exist several ways to calculate this correlation, but the best known is the Pearson product-moment Correlation explained next.

3.2 Pearson product-moment correlation

To find similarities between two objects, and their values, a mathematical model can be used. The Pearson product-moment correlation finds this similarity through calculating the linear relationship between object X's and object Y's preference values. It is this similarity that is used in the item based and user based recommendation methods described in the next section.

The formula below illustrates how a recommendation engine would compute the correlation between two vectors X and Y.

$$r = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{(n-1)S_X S_Y}$$

$$S_X = \sum_{i=1}^{n}(X_i - \overline{X})^2$$

Where $\overline{X}$ and $\overline{Y}$ is the mean value of respectively value, S_x and S_Y is the standard deviations for x and y respectively, and r is the calculated correlation. The correlation is a numeric value from -1 to 1, where 1 is a perfect correlation and -1 is anti-correlation. X_i denotes X's element i. Note that anti-correlation is used, as it provides useful information.

3.3 User based recommendations

User based recommendation systems are based on finding ranked users who tend to have similar interests and rate content similarly as the active user to whom we wish to make a recommendation. The idea is to compute neighbors in a multidimensional feature space near this (active) user. These neighbors are users that have the same interest(s) as the specified user. All of these neighbors together create a neighborhood. The size of the population in a neighborhood depends on if it is bound by the number of users or by a correlation threshold. In the first case a neighborhood size can be set by the number of users which is needed in each neighborhood, the highest correlated users will be included in the neighborhood. In the latter case the size is limited by the correlation threshold. That is; if a correlation threshold is set to one, then only the users which have a correlation of one will be members of the neighborhood. When the neighborhood has been determined the recommender will base its recommendation on the items that the neighbors have indicated preferences for.

To create a neighborhood the proximity of users has to be calculated. When speaking of the proximity of two users the term correlation is used, in general this simply means the co-relation between information known about these different users. The numeric value of this correlation is computed and is used to indicate the degree of similarity of one user and another user.

Once the neighborhood(s) have been calculated different algorithms can be used to calculate a predicted preference for each specific user for any specified item. This is done by using the preferred items of the neighbors and their proximity to the specific user in order to predict for this specific user their preference for these same items. Usually the neighbors with the highest correlation to this specific user have the strongest impact on the prediction of the user's preference for this item. This is further explained in the next section.

Although user based recommendation systems have been widely deployed, some weaknesses have appeared. The time it takes to generate a recommendation grows linearly with the **product** of number of users and the number of items. To calculate the neighborhood the correlation to every user needs to be calculated, and then a prediction needs to be calculated for every item in the neighborhood, this leads to scalability problems.

3.3.1 Prediction computation

Once the similarity has been calculated the prediction can take place. This is usually done by adding up the previously rated items. This can be done in several ways, but usually an average weighted sum is used. To clarify this process, a user based approach will be used in an example.

An item needs to be predicted for a specific user to see if it should be recommended. By adding up the ratings of all the neighbors that have rated that item and dividing the sum by the number of users we get the average rating. However, this would be a bad way of predicting ratings since some users might not have the same taste for this content type. That is why the users which have higher similarity with the specific user for which an item is being predicted will impact the final result more (hence the use of a weighted

sum). In our calculations the similarity servers as a weight for the rating (i.e., the actual rating is the similarity times the rating). The example below shows the average weighted sum formula.

$$P_{u,i} = \frac{\sum_{all\_similar\_items,n} S_{i,n} * R_{u,n}}{\sum_{all\_similar\_items,n} \left| S_{i,n} \right|}$$

Where $P_{u,i}$ is the Prediction for user u for item$_i$, $S_{i,n}$ is the similarity for item i with item n and $R_{u,n}$ is the rating of user u for item n.

Negative similarity will lower the final predicted rating. If there exist two users in the neighborhood which both have rated the item as 5, but with similarity -1 and 1 respectively, we would get the following result from the formula.

$$P_{u,i} \frac{-1*5+1*5}{|-1|+|1|} = 0$$

Since one of the user anti-correlates with the user u, they are seen as complete opposites; therefore an appropriate rating must really be the complete opposite of what the predicted user would rate the item, which is 0.

3.3.2 Slope one

Slope one uses the ratings of the items to predict the rating for an item. It uses the differences in ratings between items. The differences in ratings between the items are simply calculated by subtracting the ratings from each other. In Figure 3-1 the question mark indicates that item$_2$ is recommendable for user$_2$. The difference is calculated by using the ratings of user$_1$, by subtracting 4 from 2. The result from the differences between the ratings of these two items is then added to the user's item$_1$ rating. The result is that the question mark is replaced with the predicted rating 5 (3+(4-2)). In practice the differences from more than one user are used.

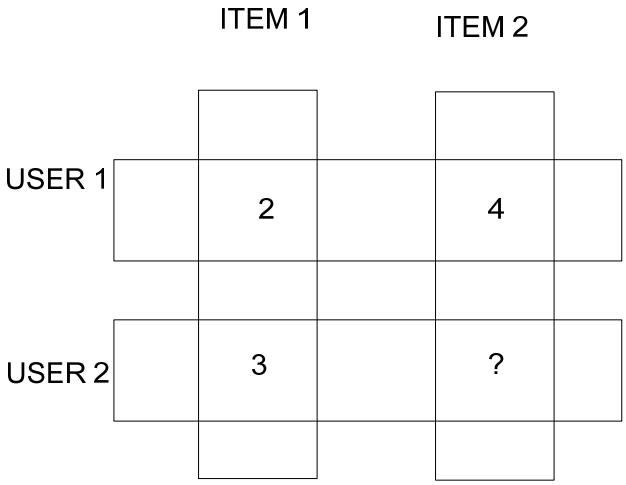


Figure 3-1: A value of preference id calculated for the user2 based on user1's similarity

The item based approach does not have the same problem of scalability as user based recommendations, since the similarities between the items are calculated for every user and can afterwards be used several times. It is of course important to recalculate the similarities when there are new purchases; in order to ensure that the new similarity information is used.

3.3.3 Clustering

Clustering is a model based collaborative filtering technique. Model based algorithms create a model from the ratings before prediction. These algorithms are often calculated offline and stored so they can be accessed several times without being recalculated.

Clustering can be used to create a subgroup of users that have similar tastes. This should not to be confused with neighborhoods. In the case of neighborhoods a user can be part of several neighborhoods and these are usually much smaller in population size that the clusters. Once clustering is done, then the techniques discussed in this chapter can be applied to the cluster, as were applied to the full data set (Thus clustering reduces the computation required, hence increases the scalability.). This is particularly interesting when the amount of data to be processed negatively impacts the performance of the recommendation engine. The cluster reduces the number of users' (rankings) that the recommendation engine needs to take into account. To make a prediction for a user the recommendation engine first looks in which cluster the user is located. Then a suitable collaborative filtering technique, user or item based, can be applied to that cluster.

Another well known model based algorithm which is used for recommendation is Bayesian networks [35].

4 Recommendation engines and libraries

Recommendation engine application programming interfaces (APIs) often allow the developer to choose which type of recommendation algorithm is used. The underlying library usually contains several recommendation techniques that the implementer can chose from, thus the programmer can easily customize the behavior of the recommendation engine. For *open source* recommendation engines it is also possible to add your own filtering techniques or modify the existing ones; rather than simply selecting one of the available ones.

A requirement in the thesis was that the programming be done in Java, which meant that the recommendation engines needed to be implemented in Java, since the recommendation engine was to be modified. Two, somewhat well known, open source recommendation engines are the Taste and Duine Toolkit (each is described in a following section). These two recommendation engines where chosen to be examined more thoroughly; as some effort had already been made by the company to examine them before the start of the thesis. Both of these recommendation engines provide the developer with tools to adapt the recommendation engines to their needs. Another well known open source recommendation engine is the Collaborative Filtering Engine (COFE) [21].

4.1 Taste

The Taste [27] architecture depends on a source of information which is used to create a *data model*. This data model will contain information about all the users, all the items, and all the preferences/ratings from the users of these items.

Taste supports Slope one and user based recommendations using the Pearson correlation and the Spearman correlation [26]. Other correlation techniques must be implemented by the developer. In the user based approach the average weighted sum is used to make predictions of items.

This architecture allows the implementer to create a recommendation engine by separately creating correlation schemes, defining neighborhoods, and creating prediction formulas; without affecting the other components. This may be of interest when adaption of only certain parts of the engine is desired.

The correlation object used is passed to the neighborhood object; and the neighborhood object and the data model are passed to the recommender object. The average weighted prediction is implemented by a method in the recommender class. Recommendations can be obtained by calling the recommender method in the recommender class.

4.2 Duine

Duine Toolkit [5], just as Taste, provides the developer with tools to create a recommendation engine. Although the architecture is a bit different from Taste, the functionality is more or less the same. However, Duine offers something that is not

available in the Taste API. This additional functionality is the ability to specify a *prediction strategy* (not be confused with the prediction techniques).

Prediction techniques are used to predict how much a user will like a particular item. The Duine recommendation engine API provides a prediction strategy which is utilized before applying a specific prediction technique. In this approach the prediction strategy predicts *which prediction technique to use*. This is desirable because some prediction techniques are better for new users that have a sparse set of preferred items, whereas other prediction techniques are better for users that have used the recommendation service for some while and have a large set of preferred items.

It is up to the developer to create the prediction strategy, therefore the developer must decide which prediction techniques to use and in which order to use them. The number of different prediction techniques the developer wants to use will probably vary from system to system. When to use which prediction technique is up to the developer to decide. Example of conditions under which to use a given technique could be: how many users prefer a particular item or how many users exist in the system. There must be a default prediction technique that is used if no conditions for the other predication techniques are met. Figure 4-1 shows an example of a prediction strategy with two prediction techniques.

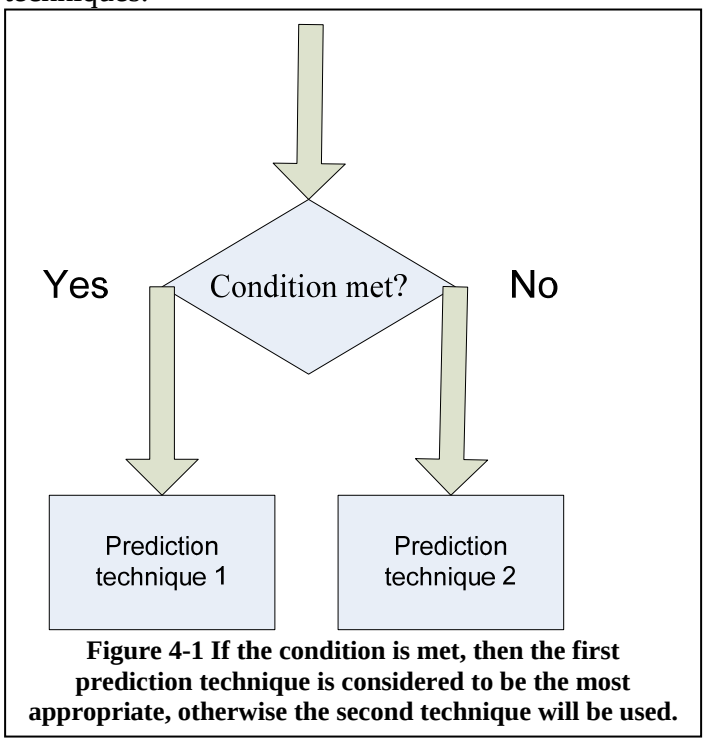


Figure 4-1 If the condition is met, then the first prediction technique is considered to be the most appropriate, otherwise the second technique will be used.

5 Semantic web

Although search engine functionality has improved, there still remain a number of limitations concerning what inputs it can understand. Thus problems occur when trying to give the engine a query in natural human language. This is something that users have gotten used to, thus users' adapt their queries to the engine's capabilities. Consider the problem of searching for a camera. When searching for a "camera in the price range from 150 to 170 euro" you will certainly get hits that do not match what you are looking for[1]. However, this is the language you would normally use when speaking to a person. Unfortunately, this query is not so simple for the computer to understand; hence the results may be poor.

The World Wide Web Consortium (W3C) has standardized languages that are used to describe resources on the web or to describe real life objects. The Resource Description Framework (RDF) [29] is a language that has been adopted to describe all types of resources and products, whether the entity is a mobile phone or a CD recording in a commercial store.

5.1 RDF

Describing relations in a formal way requires a well defined language. An ontology language, in artificial intelligence (AI) and web semantic terms, means a language that can formally describe relationships between different entities.

RDF is designed to be read by computers and is written in XML. RDF enables us to describe resources using a common well-defined syntax, so that the description can be understood independent of a particular operating system or computer architecture. An RDF description is built by combining the rules *resource*, *property*, and *property value*. The resource is **what** is going to be addressed and consists of an URI to point to the location where the resource can be found. The resource can be described by adding properties and property values to the RDF. To describe the functionality of a resource several properties and property values can be added. The example below is one possible description of a movie:

Table 5-1. Example RDF description of a movie

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://www.exampleshop.com/film#">
<rdf:Description
 rdf:about="http://www.exampleshop.com/film/Citizen Kane">
  <film:director>Orson Welles</film:director>
  <film:country>USA</film:country>
  <film:year>1941</film:year>
</rdf:Description>
</rdf:RDF>
```

[1] The result varies depending on which search engine is used. Additionally, the user's geographic location or network address can also affect the result.

Here we see that the movie can be bought from “exampleshop” and it is an Orson Welles directed film from the USA with the title “Citizen Cane” from 1941. While only three properties were added in this example, a large number of properties could be added to the description.

5.1.1 Friend Of A Friend (FOAF)

The semantic web effort has resulted in standards where ambiguity is minimized; this facilitates both machine processing of the descriptions and makes it easier for users to find what they are looking for. An example of an existing standard that makes use of this is the Friend-Of-A-Friend (FOAF) project [8]. This standard focuses on providing a standard description of persons and their relationships to other persons. This description is in the form of a RDF file which could easily be published on a website or elsewhere, in order to make the information either publically accessible or to make it available to others members of a closed community. Some of the information that can be given in the file is the user’s name, email address, homepage, and picture. This makes it much easier to find the homepages or blogs of persons, along with information about them. In the FOAF file the user lists the names of people they know and information about them that distinguishes them from other users that also have FOAF files. The FOAF standard also lets the user indicate which type of relationship he or she has with this person. For example, they can be friends or co-workers. This relationship information can be useful in applications that use the FOAF files to extract a social network.

An example of content in FOAF format, that was automatically generated [2], is shown in Table 5-2. Here we see information about the user, whom in this case is myself. It also shows a relation to “friend1” and the relation described is “knows”. To distinguish this person from another person with the same name, we have the “foaf:mbox_sha1sum” tag which contains a SHA1 hash of this person's email address, thus offering some privacy for this other user - while still allowing the system to uniquely distinguish this person from another of the same name[2]).

[2] Note that it does not actually ensure the privacy of this other person, since anyone can compute the SHA1 hash – thus all someone has to do is buy/rent/obtain a large collection of e-mail addresses, compute the SHA1 hashes and now you have an index that will let you map (bi-directionally) between SHA1 hashes and the user’s e-mail addresses! (GQMJr)

Table 5-2. FOAF example in RDF

```
<rdf:RDF
      xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
      xmlns:rdfs=http://www.w3.org/2000/01/rdf-schema#
      xmlns:foaf=http://xmlns.com/foaf/0.1/
      xmlns:admin="http://webns.net/mvcb/">
<foaf:PersonalProfileDocument rdf:about="">
  <foaf:maker rdf:resource="#me"/>
  <foaf:primaryTopic rdf:resource="#me"/>
  <admin:generatorAgent rdf:resource="http://www.ldodds.com/foaf/foaf-
a-matic"/>
  <admin:errorReportsTo rdf:resource="mailto:leigh@ldodds.com"/>
</foaf:PersonalProfileDocument>
<foaf:Person rdf:ID="me">
<foaf:name>Michel Shahan</foaf:name>
<foaf:title>Mr</foaf:title>
<foaf:givenname>Michel</foaf:givenname>
<foaf:family_name>Shahan</foaf:family_name>
<foaf:nick>Michel</foaf:nick>
<foaf:mbox_sha1sum>922f5b3e70a6f4400eba19196d412d78decbaa1a
</foaf:mbox_sha1sum>
<foaf:schoolHomepage rdf:resource="www.kth.se"/>
<foaf:knows>
<foaf:Person>
<foaf:name>friend1</foaf:name>
<foaf:mbox_sha1sum>b812ea410cd2806079b7aa1f76f9924fcb3b1307
</foaf:mbox_sha1sum></foaf:Person></foaf:knows>
</foaf:Person>
</rdf:RDF>
```

5.1.2 Triple stores

Triple stores are databases that are designed for storing, manipulating, and querying RDF data. The data from the triple stores are obtained by either query them using a query language such as SPARQL [18], or to use the API provided with the database. The name "triple store" comes from the fact that binary relations are represented as triples. For instance, the relation “Johnny likes dogs” would be represented with the triplet (Johnny, likes, dogs) in the database. Two well known open source triple stores are Jena [12] and Sesame [10].

Figure 5-1shows how the triples are usually used. The sources are RDFs which can be read by the applications through the API and stored in the database, all interaction the application does is through the API (for web applications SPARQL is often used).

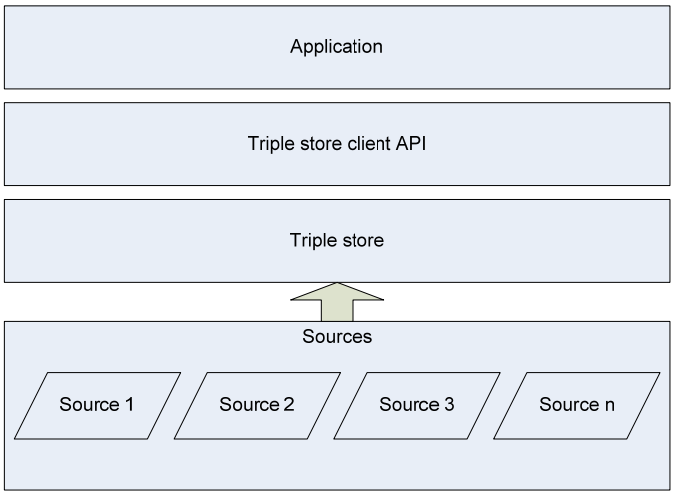


Figure 5-1 A general view of how a triple store works. Applications access the data of the database through the triple store's client API. The triple store contains data from one or more sources.

5.1.3 XFN

Xhtml Friends Network (XNF) [7] provides a way to represent relations between humans using hyperlinks. By adding a *rel* (relation) to a hyperlink, further information about your relation to the owner of the specified URL is given. One advantage is that it is very easy to understand and implement, for people that are acquainted with the HTML language. Thus XNF could become widespread. XFN provides different profiles that the user can use to denote the relationship that best clarifies their relationship. By adding "me" to the attribute the link indicates that the site that the URL is pointing to belongs to the same user as the page that the link is currently in.

The example below shows that the owner of the site "href="www.example.com/~friend1" is "friend1" and is a friend that I have met at least once, and who is also a co-worker. This example XNF hyperlink was automatically generated [22].

Table 5-3. XNF example

```
<a href=www.example.com/~friend1
   rel="friend met co- worker">friend1</a>
```

5.2 Contributions of the Semantic Web

The increase of Internet communities can help create the semantic web, as these users are more involved in what is put on the web. For commercial companies it can be a great help to post content along with a semantic description, as their products might be easier to find when searched for. The numbers of web services which utilize the semantic web steadily increases and there is also the talk of the semantic web being the foundation for web 3.0 services [24]. This also includes the social network applications that are growing through the use of semantic technologies, such as FOAF and XNF.

6 Social Networks

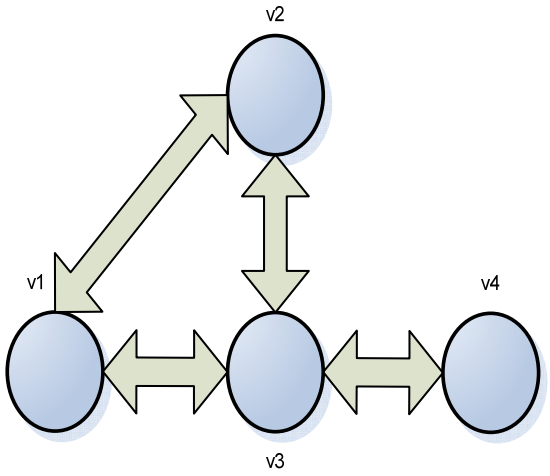
Social communities on the web and blogs are increasingly popular among regular Internet users, as they provide an easy way of expressing opinions and foster interacting with other people that have similar interests. Facebook [31] and LinkedIn [30] are only two of many communities where the user interacts with other users. Tracing this interaction, within the communities, between users can be used to compute the relationship(s) between different users. Blogs can also be very useful in understanding a social network, as bloggers often discuss specific topics with each other and blog rolls[3] have been created for this purpose. These blog rolls are used to aggregate the blogs of several users that often discuss the same topics. Hence these blog rolls can also be used to infer relationships between users.

6.1 User relations

The relationships between users are typically illustrated as a graph. In this graph, users are represented by vertices (V) and relations are represented by edges (E). Each user can have one or more relations to (other) users, as depicted in Figure 6-1. In a social network the users usually do not have a relation to themselves. One thing to notice is that the edges in the figure are bidirectional, as this is how relations generally exist in practice. However, this might not be the case when extracting the data automatically from different sources. One user may feel that he/she has a strong friendship with another person, while the other person does not have this perception of the relationship. This can also occur when only one of the users has input their relationship information, thus the other users' relationships are not known, but some might be inferred -- if one can assume that the relationships are bidirectional. The direction of the relationship will be discussed in section 7 where trusts in relationships are explained.

There might also be additional information annotating the edges of the graph to indicate how close the relation is, thus creating a weighted graph. A larger weight could indicate that the relationship is very strong. If the edges where weighted this could be presented using numbers instead of Xs in the matrix representation of the graph in Figure 6-1, which indicate the edge value.

[3] Blog rolls – is a listing of websites on a blog which are often linked to other blogs or interests.



	v1	v2	v3	v4
v1		X	X	
v2	X		X	
v3	X	X		X
v4			X	

Figure 6-1: On the left: The figure presents relation between different users, denoted by vertices. The edges represent whom the users have relation with. On the right: the same users and relations presented as a matrix. Note that bi-directional graphs have a symmetric matrix equivalent.

6.1.1 Graph properties

If the graph is not weighted, then the strength of relations might have to be inferred. This could be done by looking at how the groups are formed and how many common friends' pairs of users have. This section will focus on graph properties and examine how they could be used to predict relationships. Note that these inferred weights can be used to annotate the graphs.

6.1.1.1 Cluster coefficient

The occurrences of social groups in real life are very common. From a graphical perspective, groups are presented as clusters. The relationships that clusters have are called *bridges*, and are usually based upon relations between a few persons in one cluster to a few persons in another cluster. (This means that the social network represented by a graph is often not a tree, as the social network may have loops.) The clustering for single vertices can be calculated using the information about the relationships of neighboring vertices; this calculated result is called a *clustering coefficient*. This coefficient is the ratio of the number of edges that *exist* between the neighbors to the number of *possible* edges between the neighbors.

6.1.1.2 Geodesic

Stanley Milgram's study of social networks focused on how transitive relations exist in practice [13]. The study showed that a randomly selected person in Boston and Omaha (both in the United States) could each reach a specific person by mail, by simply sending it to someone they were already acquainted with, who would re-send it to someone that knows a person closer to the target, etc. The results of his study showed that the average depth, i.e., how many persons were needed on average, before reaching the target was 6. The smallest depth between two vertices in a graph is called *geodesic*, representing the shortest path between them.

6.1.1.3 Clique

A *complete graph* is a graph where all the vertices have connections to each other. In a larger graph it might be interesting to find subgroups which form a complete graph. A graph can contain several subgroups which are complete, and finding out how many, the biggest, and which they are is called the *clique problem* (finding the maximum cliques). Unfortunately the clique problem is NP-complete. As the clique problem is a special case of the *k-plex problem,* which finds the maximal sub-graph in which every node in the sub-graph is adjacent to at least $g_s - k$ nodes, where g_s is the number of nodes in the sub-graph [23].

6.1.1.4 Core-Periphery structure

A Core-Periphery (C/P) structure is very common in social networks and consists of two subgroups: the core subgroup and the periphery subgroup. The nodes in the core subgroup have densely connected ties with each other, whereas the peripheries are more loosely tied [24].

6.2 Social network extraction

Social network extraction attempts to extract information from a source in order to derive a social network. What source is used is depends upon what the objective of the network is or what limitations exist. One could for instance use a survey to extract the information about who is a friend of whom and to what extend they spend time with each other. This approach would be explicit and would require the users to fill in the form. Unfortunately, few people are eager to do this. Another approach is based upon mining information from multiple sources, either once or at certain time intervals, i.e., extracting information that has been previously stored by the user or a third party.

The choice of source or sources from which you extract the information depends upon what information you have access to and what information you need. It also depends on what legal agreements can be made with the end users or a third party. While the whole Internet is potentially available, the system that extracts the information might be limited to certain sites on the Internet or to private networks, such as companies and universities. A developer can choose whether to extract the information from a local source, such as a company, or a global source like the Internet. We next consider these two scopes for information sources.

6.2.1 Local

As previously mentioned a local resource is often hidden from the world and is only visible to the persons within the local community or the administrators of the community. Email is a resource that in many cases is non-public (for reasons of privacy and/or company policy). However, e-mail messages contain a great deal of information about the relationships between users. By looking at how many messages have been sent between different users and who the recipient is, a (possible) relationship between the people can be inferred. Unfortunately, this does not say what relationship they *actually* have. Additional information can be gathered by processing the contents of the messages

(for example, by using an artificial intelligence based engine to parse and analyze these messages) in order to infer the type of relationship between the users.

Web communities reveal information about users and their relation to other members of the community. However, exactly what information is available depends upon what kinds of services are available. It is common to have functions to add friends or acquaintances - along with their instant messaging address to your social network. Someone who has access to this information can examine the social network, and depending on the deployed services, the model of relationships can be very accurate.

These are only few of the many sources that a system could gather information from. Depending on the location of the information different extractor systems may be needed extract the most out of the source. (See section 6.3)

6.2.2 Global

The trends of web communities, homepages, and blogs have increased the amount of information about people that is publically accessible on the Internet. Data mining this information can yield sufficient information to determine relations between people. There are several ways of doing this, but they have different degrees of success. These methods either make use of the semantic web, as discussed previously, or use an intelligent system that can gather and analyze the information from regular text on the Internet using a technique called *co-occurrence*.

Co-occurrence uses search engines to see if two or more names occur on the same website or within web pages of the same domain; if so, then persons with these names might have some relationship. Searching for two names on the internet would probably not immediately yield a useful result, but applications that use co-occurrence often add keywords to the search in order to limit the search results and increase its accuracy. An application can limit the results even more by restricting the search to specific domains, which localize the data, yielding even greater accuracy. The closeness[4] between the persons is determined based upon the number of hits the search engine finds. Some of the existing co-occurrence systems [25] [14], such as the one to be introduced in section 6.4, can narrow their search to co-occurrence to, for example, only researchers, writers, companies, etc. Because the complexity of the system decreases when the system narrows the types of persons to search for, there is an advantage to using narrower terms as the keywords. A narrow term is simply a more specific terms for what you are search for. For example, the keyword “ant” is narrower than “insect”.

6.3 Social Network Extractors

Some of the social network extractors that exist make use of **both** global and local recourses. This section will describe two well known extractors; where one of them is open source software and the other is not.

[4] Closeness – How close persons are is determined by the co-occurrence of persons. Closeness is determined by the number of co-occurrences in the search result and the context in which they occur.

6.4 Flink/Elmo

The Flink system [14], developed by Peter Mika, gives users the possibility to use both global and local resources. It is specially designed for finding researchers' relations, but could work with other people as well. It is a combination of the open source system Elmo [11] and the commercial Flink. An overview of the architecture of the Flink system is illustrated in Figure 6-2.

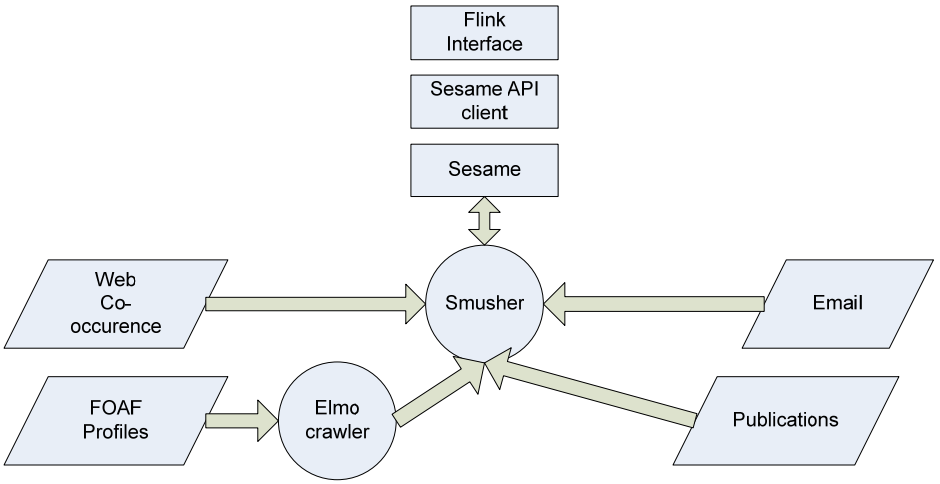


Figure 6-2: The architecture of the Flink system, illustrates how the different sources are aggregated and passed to a database.

The figure shows that this system makes use of co-occurrence, FOAF profiles, emails, and publications. The FOAF files are followed using a crawler, which follows the URIs to find other FOAF files. It also uses co-occurrence, searches publications using Google Scholar, and parses email messages which are downloaded from mail servers using POP3 or IMAP. The *smusher* aggregates all the information gathered from the different sources. This extractor is a good example of using different kinds of techniques to gather information. It aggregates and stores the results in the Sesame database (see section 5.1.2).

Note that Elmo can be used without Flink, but some of the functionality will disappear. For more information about Elmo see [11].

6.5 Google Social Graph API

Google provides a way for applications to search for relations using the semantics of the web. More exactly it searches for the XNF links and FOAF files that are spread all over the web. By entering a URL with the correct parameters, the Google social graph searches for relations and returns them in JSON format [16]. The example below shows how a URL might look like when searching for the relation of URL “a”. In this example the result will show only the edges that are pointing in to “a”, as specified by “edi=1”, the edges that point to the “a” URL will be omitted, if not specified. The first line shows the general format, while the second shows a specific example.

Table 6-1. Google Social Graph example

http://socialgraph.apis.google.com/lookup?<parameter 1>&<parameter 2>&<parameter n> http://socialgraph.apis.google.com/lookup?q=A&edo=1&edi=1&pretty=1

7 Recommendation engines and trust

This chapter will examine some features of existing recommendation engines which utilize the users' social network. Most of these are *trust based systems* which use information from users about whom they trust. Some general properties of trust are that trust is transitive and asymmetric, which shows that trust is not bidirectional, and will therefore produce a directed graph. These trust systems can in some cases outperform regular collaborative filtering. They have proven to be very accurate when users deviate from the general public's opinion [36], i.e., when users are not near the general population norms. Studies have also shown a correlation between user similarity and trust in real life [37].

Jennifer Golbeck, et. al. [36] states that transitive trust is not "perfectly transitive", meaning; if A trust B and B trust C, then it follows that A should trust C. But as this not a direct trust, the degree of trust will decrease, and will continue to decrease as the distance from user A increases. To determine how much users trust one another the trust relations needs to be inferred. Note that trust is on a personal level. That is; the values that are inferred for every user-user pair which exist in the network, this is also referred to as local trust.

Global trust refers to trust which is calculated for a user and is the global trust opinion. The local trust is considered more appropriate for recommendations, as the recommendations are also on such a personal level [38].

There are a number of trust based systems, we will consider two of them which are well known. TidalTrust [38] uses an average weighted prediction as presented in section 3.3.1 to predict ratings for items when doing content filtering. EigenTrust[39] uses a PageRank algorithm [40] to infer trust in a peer-to-peer system, but is not applicable to regular social networks because EigenTrust uses performance as a measure of trust, which does not differ so much from one peer to another, whereas social network uses humans, where opinions about a person's trust can differ much more [38]. These two systems utilize trust to make rating predictions in order to make recommendations of items. This thesis differs from other trust systems by looking at social graph topologies which indicate that there exists a strong relationship between users. These graph properties and features will be taken into account when creating a recommendation engine to exploit information from a social network.

8 User experience

An *accurate* recommender does not necessarily imply that it will be the *best* recommender. If a user buys the movie “The Lord of the Rings: The Fellowship of the Ring”, what type of recommendations would be suitable for this user? One recommender might argue that the two subsequent movies in the trilogy are good recommendations, while another recommender may consider that that it is obvious that the user knows about them already, thus it is better to recommend something unrelated.

It is important to separate an *accurate* recommendation from a *good* recommendation. A *good* recommendation takes serendipity into account as well, whereas an *accurate* recommendation might simply lead the user to a “similarity hole” [24]. A *similarity hole* occurs when the recommended items are very similar to what the user has already purchased. In this case, buying the recommended item will only lead the user deeper into this hole. This problem is most common for new users that have input few preferences, thus the recommendation engine has very little to base its decision on.

Friends do not necessarily have the same tastes as you, which might imply that there is little value for recommendation systems based on social networks. However, the goal of this thesis is to explicitly improve the recommendation system's results by using information from social networks, therefore an important metric to evaluate the performance of the proposed system is that it should not *decrease* the appropriateness of the recommendations. Instead there should be additional positive **values** for the users that chose to use the service and actively make use of it.

Unfortunately, there is no easy way to determine if the recommendations that are given to the users are accurate. Usually a recommendation system makes use of *evaluators* which predict the purchase rates for items that they already know the purchase rate for. If the probability of an actual purchase of the item is close to the predicted rate, then the prediction and accuracy are assumed to be good. When working with recommendations based upon social networks, the predicted accuracy can be hard to evaluate in the same way as recommendation engines would usually do. Instead a simple means to determine the accuracy would be to base the evaluation on test-users who explicitly give their opinions on the quality of the recommendation.

9 Extracting social networks

Extracting a social network from a source is done to determine the friends that a user has. This begins by choosing a source where the information could be gathered. The number of sources is limited to one in this thesis, as a practical limitation for this thesis, due to time constrains. Ideally the extractor could be designed to collect information from several sources and to merge them; as explained in the future work section.

9.1 Social network source decision

Sources such as communities, Call Data Record (CDR[5]), mailing lists, etc. might contain a huge amount of data. It is important to know how much of this data is relevant for the inference process and also if this data is reliable. There are many communities where it is possible to add friends. This would seem to be directly relevant for the social network extraction, but internal Ericsson studies have shown that people tend to accept contact or friend requests to avoid being perceived as impolite, which makes this data unreliable.

Call data records contain information about received and made calls and the duration of the calls. However, they do not say anything about the relation of the caller and callee. The calls can be strictly work related and the two parties may have no other common interests, thus this data could also be unreliable.

This thesis has so far been very general about the content which is to be recommended, as the same technical approaches can be applied to many types of content. However, depending on the content it might be good to have different data sources. Additionally, since different communities have different purposes, people do not have same friends on LinkedIn and Facebook; hence the content recommendation will differ for this reason as well. LinkedIn being more business oriented is more appropriate for content recommendations which are business oriented, whereas Facebook is used primarily to keep in touch with friends; hence for content recommendations which you would share with your friends, Facebook is probably more appropriate source. In this thesis, we have decided to use Facebook as a source as it contains contacts which the project hopes to use to provide personalized TV recommendations; where the user is going to view this content for their personal enjoyment during their non-working hours.

9.2 Extracting social networks from sources

To extract data from sources such as communities the administrators of this social network must first make the data accessible through an API or deliver it in some other way. Social networks normally have strict rules regarding that can access this data, how to access this data, and what can be done with the data. The license agreement of the corresponding community must be read carefully and followed.

[5] CDR – The call data record contains detailed information about calls, such as start and end times, calling number, and called number. This information is generally used for billing purposes. These records are stored in a database connected to the telephone operator's network.

9.2.1 User input

The user's experience often will depend on *how much* the user has to interact with the system. The idea is to automate those things that are possible to automate, if this will result in improved results. However, there might be some things that the user explicitly needs to do to get the system working well or to get the system working at all. It is important to clarify what each user must do, as the specific needs for user input can affect the system design. Additionally we must decide how much accuracy is required and how automatic the system is expected to be.

9.2.2 Accepting friends as content recommenders

Once the social network graph has been created the overall view of the user's social network is known. Yet the user might want to separate their friends into those that have good taste in movies from friends with less good taste. This refinement of the graph is also necessary as users may have people in their community that they do not know well or do not even know at all, as explained in section 9.1.

The system should support several different content types. For each content type a new graph will be created with different connections between the users. These *per content type graphs* are subsets of the entire graph which was extracted. The user can chose not to have any relation to a specific user by not accepting the person as a potential recommender for a specific content type. The friends associated with the user will be a (per content type) subset of those that exist in his social network.

By trusting another user to be a friend you add an edge between that user and yourself. This edge is directed. As users chose the list of people they trust as recommenders for a specific content type the (sub-)graph will grow. One (sub-)graph will be created for each content type. For the purpose of this thesis we will examine the case of one content type, thus we will compute one sub-graph.

10 Building Graphs

As the social network will be processed as a graph, it is convenient to use existing tools for manipulating graphs.

10.1 JUNG

Java Universal Network/Graph Framework (JUNG) [28] is an open source library for creating, visualizing, and analyzing graphs in Java. It provides several ways of reading and creating graphs. It has a well developed API for visualizing graphs using several different techniques.

JUNG has several algorithms which can be used to calculate well known graph results. The implementation of JUNG makes it a good tool for social networks analysis as it stores vertices that are sparse in an efficient way, reducing the required memory space, in comparison to other graph tools which grow as $O(n^2)$ with the number of vertices. This is especially important in this project as the number of people the user will need to add as recommenders may be very few, in comparison to the number of users in the system.

A drawback is that there is no easy way of creating/defining your own vertex or edge. The release that was investigated was the stable release 1.7.6, the alpha release of JUNG2, that exists at the time of writing this thesis, has not been thoroughly looked at. The drawbacks in release 1.7.6 may have been fixed for the JUNG2 release.

10.2 NetDraw

Is a graph visualization application which allows the user to visualize graphs which exist in a certain format, such as Pajek[6]. But it does not provide a way to integrate the code into an application. According to their website[7] NetDraw should be able to handle 10000 nodes on a computer with 2GB of memory if the data is sparse. This is not very much in comparison to the other tools which are discussed here.

10.3 JGraphT

JGraphT [8] is Java graph library which is released under the GNU Lesser General Public License. It supports most functions & types of regular graphs, such as directed and undirected, with weights or without. In addition to this the library also provides an ability to visualize the graphs, which turned **not** provide the possibility to interactively create and modify graphs from the visualization view. This is a drawback since it was desired that the implementation would also be used in demonstrations. However, it is scalable and has support to handle a few million vertices and edges in a graph. It is optimized for data modeling and study of algorithms. JGraphT seemed to be a good candidate, since it provided a Java API and most of the desired functions existed.

[6] For more information about the software and the graph description format Pajek see http://vlado.fmf.uni-lj.si/pub/networks/pajek/

[7] http://www.analytictech.com/Netdraw/netdraw.htm

[8] http://www.jgrapht.org/

10.4 Pacific Northwest Lab (PNL) Starlight

Starlight[9] is a visualization tool for analysis of statistical models. It provides advanced visualization in 3D. Input of data is in XML format and it has an API through which the functions can be called. It is also a commercial product, which is a drawback since there were limited resources available for this thesis project.

10.5 Choice of tool

For the purpose of this thesis, we have chosen to use JUNG because it has good documentation, a Java API, good visualization options, and many of the functions which were thought to be needed at that time existed in the library. As it is used by many developers a large amount of information could be found in the forum [10]. JGraphT was also an option for the implementation, but due to its limited visualization functions it was no selected.

[9] http://starlight.pnl.gov/
[10] https://sourceforge.net/forum/?group_id=73840

11 Inferring relations

Social network analysis has shown that friends tend to form clusters, where the people in the cluster have very close relationships. Strong ties are often tightly clustered; whereas weaker ties are weakly clustered. Clusters are often connected to each other by a few persons in each cluster (bridges as described in section 12.2.1). To find close relationships and distinguish them from other relationships a weighting system can be used. Closer friends have a high value, less close friends have a lower value, and a friend of a friend has the lowest value. How these values vary depends on information extracted from the social network and what the network model is.

11.1 Finding close friends

To find close friends of a user, information about this user's neighbors' relations needs to be gathered and processed. If there is a neighbor who also is a friend of other neighbors, then the three users together form a relationship in which all three know each other. These types of relationships indicate that they have strong ties to each other. The sub-graph where every person is a friend who knows the others is known as a clique in graph theory terms (see section 6.1.1.3).

As the graphs which the system will be working with are directed graphs, a special case of the clique applies. The definition of a clique in this case is that if the active vertex has two or more neighbors which he or she trusts **and** if they have a relation in either direction of the graph; then this graph forms a clique. Figure 11-1 shows an example of such a clique, when the active vertex is V0. This can be implemented with minor changes in an existing clique finding algorithm.

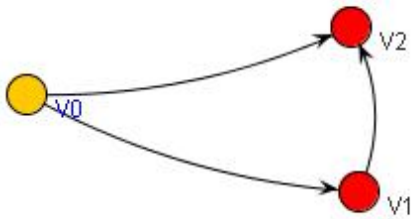


Figure 11-1: The figure shows a clique which is formed using directed edges. The active vertex here is V0.

11.1.1 Non-deterministic Polynomial time

To find cliques in a graph is considered to be very computationally expensive, as the computation time to find cliques in a graph is known to be a non-deterministic polynomial time problem (NP-problem). This means that for any algorithm the time complexity grows exponentially as the graph grows. In polynomial time problems

(P-problems), the problem can be solved in polynomial time proportional to the size of the graph.

In either P or NP problem, by reducing the amount of data that is going to be analyzed the problem can be solved within a reasonable amount of time. The graph which will be processed will be a subset of the entire graph, choosing only the users within a certain distance from the active user, as described in section 11.3. Another common approach is to approximate the result, thus finding a result that can be calculated quickly, but that is sufficiently close to a solution to the problem that this solution is acceptable.

11.2 Finding Transitive relationships

Transitive relationships are relationships which have a distance from the active user that is larger than one. Users that have a relationship to the active user through another friend are not expected to have as close a relation to that person as users that are direct neighbors.

As the distance increases the strength of the relationship decreases, therefore it is appropriate to limit the depth of how far to search for relationships. The number of mutual friends will also influence the result of the relation. In this case the transitive relationship is stronger than when only one mutual friend exists. Figure 11-2 compares two vertices which both have a distance of two from the active vertex, but one of the vertices is considered to have a stronger relationship due to the number of mutual friends it has with the active vertex V0.

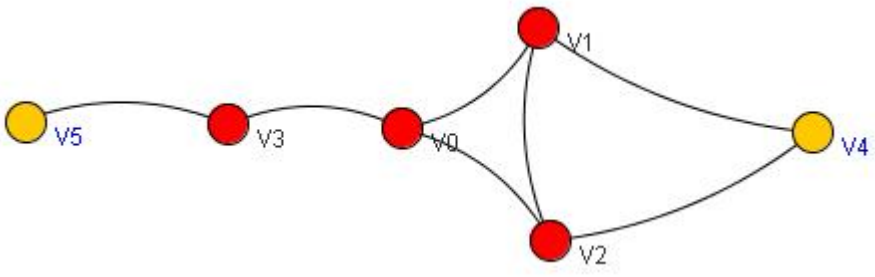


Figure 11-2: Two vertices (V4 and V5) with distance two from the active vertex V0. V4 is considered to have a stringer relation to the active vertex (V0) as V0 and V4 have two mutual friends, whereas V5 and V0 only share one mutual friend. Note that direction on edges has been omitted for simplicity.

11.2.1 Determining Depth

As mentioned above the strength of the relation decreases as the distance from the active user increases, thus bounding the search depth improves run-time performance, as less computation will be required. The value of the distance should be small, as the small world problem reveals that large depths can leads to persons with which you have no relationship at all. With this in mind a depth of one would not utilize the transitive relationship at all; while a depth of two would include the friends of friends. The depth of three seems a practical distance bound, but our analysis could be further constrained to

shorter distances - as explained in section 12.3.1.4. The depth between users will be determined by computing the Dijkstra Distance, using Dijkstra's shortest path algorithm, with the starting node being the active user.

When the active user has a large number of neighbors the time for inferring relationships will increase as explained in section 11.1.1. If the desired number of recommenders is known, then the depth can be set dynamically. In our computations, the depth bound of the sub-graph will vary depending on the need for greater depth or not.

Figure 11-3 shows vertex V0 with its surrounding vertices. If the desired number of recommenders is larger than four, then the depth of two will be used when calculating rankings. If four or fewer recommenders are desired, then the depth will be bounded to one as the direct neighbors are sufficient to calculate these recommendations. As it was assumed that larger distances have less strength; therefore there is no point of checking at larger distances.

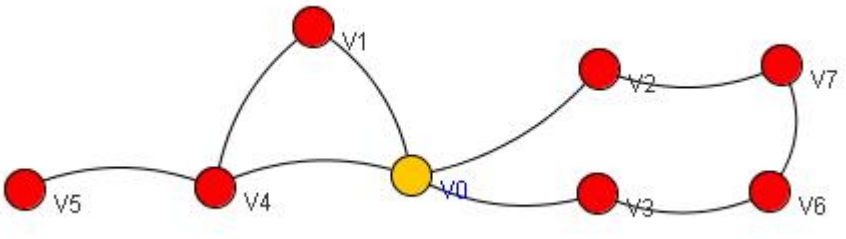


Figure 11-3: The active vertex V0 with maximum distance of two to other vertices. Note that direction on edges has been omitted for simplicity.

11.3 Calculating relation ranks for users

For each user in the graph there will be values indicating how strong their relationship is to other users. Theses values are calculated based on the properties explained above. Associated with each user will be a database table containing these values (i.e., the strength of relationships with other users). As each user will not have a relationship with all of the users, the database table will also be limited to only those with a strong relationship to the user (for example, users a maximum distance of 3 from the user). Such a table is shown in Table 3-1.

Table 11-1. The inferred relations for User A. The ranks are shown as a decimal value on the right.

User A	User B	0.8
User A	User C	1
User A	User F	0.5

Each vertex, within the maximum distance from the active user, will start with a default value. This default value will be determined by the distance from the active user. The default values of the vertices are determined by this formula:

$$D_i = \max\_depth - depth_i$$

Where D_i is the distance to vertex i, max_depth is the maximum depth the algorithm will take into account, and $depth_i$ is the depth of vertex i from the active user. Additional values can be added to this default value, based how the vertices are connected, as described in this section.

11.3.1 Dividing large cliques

All cliques larger than three can be divided into cliques of size three. This means that all cliques can be presented as a set of cliques of size three. The formula that calculates how many sub-cliques exist in a large clique is bounded by the following formula: $\frac{n\times(n-1)}{2}$. For example the clique of size 4 could be divided into 6, $\left(\frac{4\times3}{2}\right)$, cliques of size three. A person with 2 cliques, one of size 3 and one of size 4 would have a combined clique of size 9. This way we can compare different users with respect to *how many* and *how big* their cliques are.

11.3.2 Alternative to calculating and dividing cliques

When dividing large cliques; information about the size of the clique is lost. The information that is stored is simply how many sub-cliques exist in the graph that the specified vertex is member of. If this information is not used, then there exists an alternative way of predicting the relationship with the active user.

The idea is to find the cliques of an active vertex to predict a relationship. By dividing the cliques as described in section 11.3.1 we find all the smaller cliques. Of these, only the divided cliques that have the active vertex as a member will be used. Figure 11-4 shows a scenario where cliques are calculated based on the active user V0 as a member of the cliques. The user V6 is a member of three cliques ({V6, V7, V0}, {V6, V4, V0}, {V6, V5, V0}), these are all of size three.

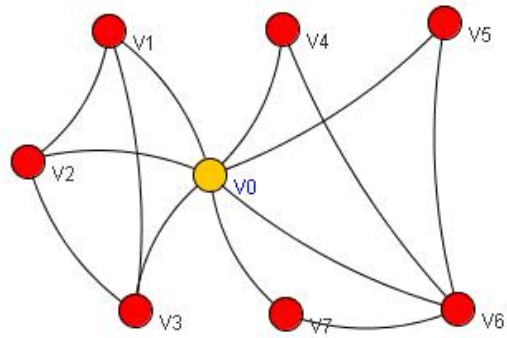


Figure 11-4: The graph shows how V6 has more cliques than any of V1, V2, and V3 although the clique {V0, V1, V2, V3} is larger than any of the other cliques. Note that direction on edges has been omitted for simplicity.

If the clique {V0, V1, V2, V3} is divided into smaller cliques, all must still have V0 as a member, thus it would be split up into {V0, V1, V2}, {V0, V1, V3}, and {V0, V2, V3}

(note that {V1, V2, V3} does not have V0 as a member and is therefore not used). We see that the vertices are members of two cliques each, except for V0. This means that there is a loss of one clique in this clique division. This will lead to different result between the regular clique counting algorithms, but will give an approximation.

As the number of cliques that the vertices are members of is related to the total number of in and out edges of the vertex, the number of cliques can be easily computed by subtracting one from the number of in and out edges. For example, V2 has 3 edges and is therefore is a member of 2 cliques to whom V0 is also a member of.

11.3.2.1 K-nearest neighborhood

By using the K-nearest neighbor algorithm a computation similar to clique division can be made **without** calculating the cliques. This algorithm extracts a sub-graph bounded by a maximum distance from a specified vertex. Figure 11-5 shows the users which have a maximum distance of one from the active vertex V0. All other vertices will be discarded from the calculation. All edges which are connected to this distance one sub-graph will be discarded. This will give the same figure as shown in Figure 11-4.

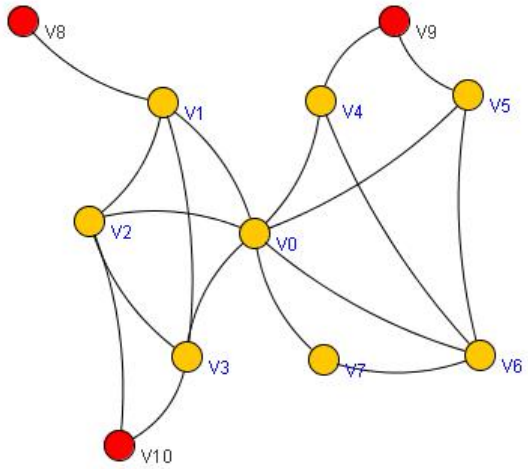


Figure 11-5: The yellow marked vertices have all maximum distance one from the active vertex V0. Note that direction on edges has been omitted for simplicity.

With this newly generated graph we simply count the edges and subtract one, as explained in previous section. This gives V6 a value of 3 (= 4-1), the same result as obtained by clique calculation and division. For each of the vertices in the clique {V0, V1, V2, and V3} this gives the result 2, which is also the same result as from the clique division.

11.3.3 Inferring relationships using clique

Tightly clustered relations are indication of strong ties [34]. Since a clique is a cluster which is fully connected, they are indications of good ties. The active user might be part of several cliques, with different users and sizes. A bigger clique means more mutual friends. Therefore, the users that have a bigger clique have higher rank values than users in smaller cliques.

A neighbor of the active user might exist that is a member of more than one clique with this active user. This leads to more mutual neighbors, hence will lead to a higher rank. To calculate this we must know what cliques exist and which users they consist of. Knowing this, additional information can be added to the edges to give information about the relationship to a specific user. Inferring the relationship based on how many cliques and how big the clique is a matter of determining what information is considered to be the most relevant.

The formula which is used to calculate the value to add to the default value for the active user's neighbors is:

$$user\_rank = (\max\_depth - distance) + \frac{|q|_{v,w}}{|Q|_v}$$

Where $|q|_{v,w}$ is the number of divided cliques of which vertex w and the active vertex v are members of and $|Q|_v$ is the total number of cliques that the active vertex is a member of. The division by the total number of cliques is done to give a percentage of the number of cliques the user is member of. Higher percentage means higher rank.

11.3.4 Inferring relationships using an alternative method

Using the alternative method presented in section 11.3.2, we can computer the rank by using the same formula as in the previous section, but with different definitions of $|Q|_v$ and $|q|_{v,w}$ - specifically:

$$|q|_{v,w} = nr\_of\_edges\_of\_v - 1$$

$$|Q|_v = \frac{\sum_{\forall\_neighbors\_w\_of\_v} |q|_{v,w}}{2}$$

11.3.4.1 Inferring transitive relationships

Calculating the rank for transitive friends' relationships differs from calculating the rank for direct neighbors. Users who do not have direct relations with the active user will have a less strong relationship. However, larger numbers of mutual friends will lead to higher rank. Therefore, the ranking will be partly dependent upon how many of the active user's neighbor the user knows.

As the different neighbors of the active user have different ranks, their value must be weighted in the calculation of the transitive relation. The active user is more likely to have a good relation with someone if the have good relation with a mutual friend. Also the actual edge value is important as it describes the direct relationship with the friends of a friend. We can calculate the user's rank using the formula below:

$$user\_rank = \frac{\max\_depth - depth + 1 - \left(\frac{c}{nr\_of\_mutual\_friends}\right) - (1-c)mean\_value\_neighbor\_ranks}{\max\_depth - depth + 1}$$

Here the rank increases as the number mutual friends increases and is further increased if these edge values are high. Note that the parameter c is a constant which determines the balance between the importance of the number of mutual friends and the neighbor's rank value. Figure 11-6 shows the case of two graphs with active vertex V0. On the left node V3 has three mutual friends with V0 in comparison to the case on the right where there exist only two; therefore node V3 on the left will get a higher rank.

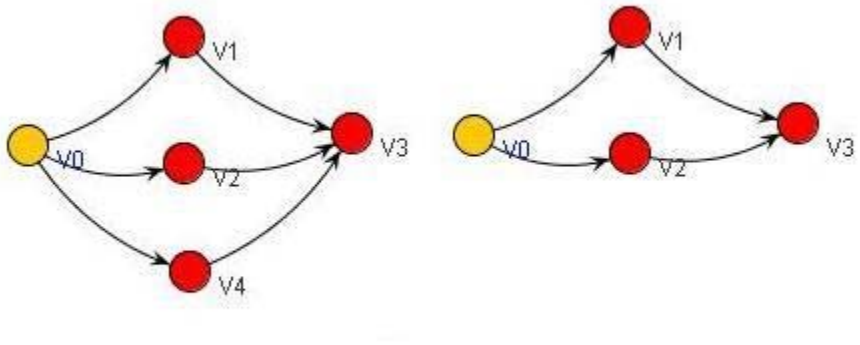


Figure 11-6: The vertex V3 on the picture to the left has more mutual friends than V3 on the right picture and will therefore have a higher rank.

12 Tests

12.1 Relationship Inference Testing

Relationship inferring is a crucial part of the project. The relationships must be calculated correct accordingly to the specifications given in the previous chapter. As the implementation of relation inferring progressed, a number of smaller tests were made. In addition to checking that the inferences were correct, we also wanted to see how the computation time changed with the size of the graph (in terms of the numbers of vertices) and the numbers of relationships (edges).

To make the testing as close to a real-life scenario as possible the graph needs to look like a real social network. However, as we did not have access to actual social networks while developing this code, we chose to implement a set of graph generators to generate test input.

12.2 Graph Generators

We initially considered using JUNG to generate social network graphs for testing. Unfortunately, the graph generators in JUNG were not suitable for generating social network like graphs, thus we developed our own generator as described in the next section.

12.2.1 Social Network Generator

To generate a social network graph for testing, a social network generator was developed. It was designed to have some of the usual characteristic of a social network. These properties are:

Clusters	The cluster property will ensure that there are parts of the graph where vertices are more tightly connected to each other, and that these are not randomly distributed.
Bridges	The bridge property ensures that the clusters are somehow connected. This is done by randomly selecting users in the clusters to connect to other randomly chosen vertices.
Small world property	The third property goes hand-in-hand with the bridge property. The small world property makes sure that every vertex can reach another vertex by a maximum of a small number of steps. In this thesis the small world property is not explicitly enforced.

The parameters required when generating a graph are the number of vertices that are to be created and a number indicating the average size of the clusters. Specifying the average size of all the clusters enables the generator to determine the number of clusters that are going to exist by simply dividing the number of vertices by the average cluster size. This number of clusters is created, with each vertex randomly assigned to one of the clusters. The vertices will be randomly connected to each other within the cluster. The number of vertices to connect to within the cluster is randomly chosen, but will be

affected by the cluster size. This procedure completes the first step in creating a social graph. The second step is detailed in the following sub-section.

12.2.2 Asset Type Graph Generator

In the relationship inference process, the graph which is inferred is a graph which provides information about which users trust what other user(s); this is used as input to a recommender for a specified asset type (in this case some type of content). This means that a directed graph needs to be generated and passed to the inference processes. This adds the extra requirement that the graph should be directed. However, this removes the requirement that the graph needs to have the small world property - as it is no longer a social graph, but an asset type graph.

12.3 Test Bed and Testing Procedure

The tests for the inference process were conducted using a computer with the Intel Core2 Due T7100 1.8 GHz CPU and with 2 GB of main memory. The amount of heap memory space allocated to the Java program during execution was set to 800 MB, as this was approximately the amount of physical memory available in the test system. The first step of the testing was to generate the graph, which allocated memory to store this graph.

12.3.1 Performance Tests

Testing was split into several different test cases; where each test case investigated the performance of inferring depending on different aspects. These tests revealed how the structure of the graph could impact the performance and how the performance differs when using different algorithms for solving the same problem. All of these tests where done on an asset type like graph (i.e. a directed graph.)

12.3.1.1 Clique Counting and Edge Counting

The main goal of the first test case was to compare the clique counting and edge counting algorithms; in order to decide which was the most efficient and to select which algorithm should be used in the final implementation. The algorithm chosen was used in the subsequent test cases.

The first test compared the two algorithms by using a small number as the cluster size. The cluster size affects the number of edges that are created; thus if one algorithm was more efficient in the case of sparse connections, but became less efficient as the number of edges increased this test should reveal this behavior.

The first tests were done with a sparse set of edges, as shown in Figure 12-1. Comparing the time it took to infer relations with different parameters showed that the clique counting algorithm had better performance. In the second test, shown in Figure 12-2, with denser edges the clique algorithm showed better results than the edge algorithm. The edge counting algorithm is called the “K-nearest” algorithm in the figure, as this was the specific algorithm used to count the edges.

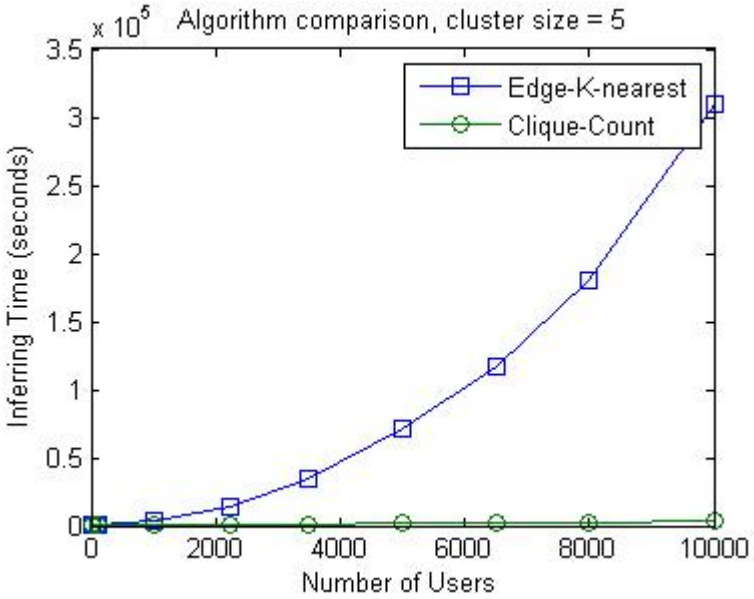


Figure 12-1: A comparison between the Edge counting algorithm which uses K-nearest and a Clique counting algorithm with an average cluster size of 5.

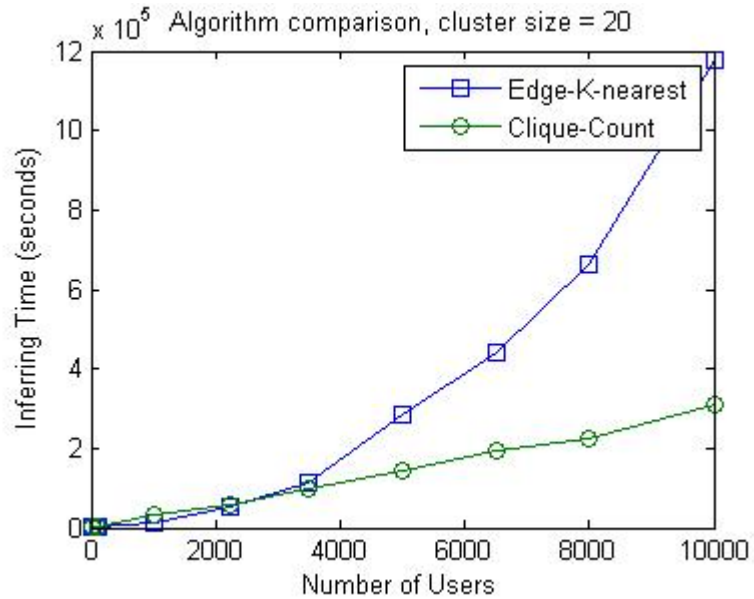


Figure 12-2: A comparison between the Edge counting algorithm which uses K-nearest and a Clique counting algorithm with an average cluster size of 20.

The edge algorithm used the K-nearest algorithm to create a sub-graph with the neighbors at a distance of one. This method is part of the JUNG library. After the test the edge algorithm was modified to replace the K-nearest algorithm without our own

solution, as the K-nearest neighbor algorithm was thought to be the source of the bad performance. A second run of the test was done, this time with our own edge counting algorithm. This time the edge count algorithm outperformed the clique counting algorithm, as shown in Figure 12-3 and Figure 12-4. As a result of these measurements, our edge count algorithm was chosen to be used in the remaining tests, despite the difference between the algorithms (as discussed in section 11.3). The edge count proved to be that much better in terms of time complexity; that although it was an approximation rather than an exact solution – this method was chosen, as it is not NP.

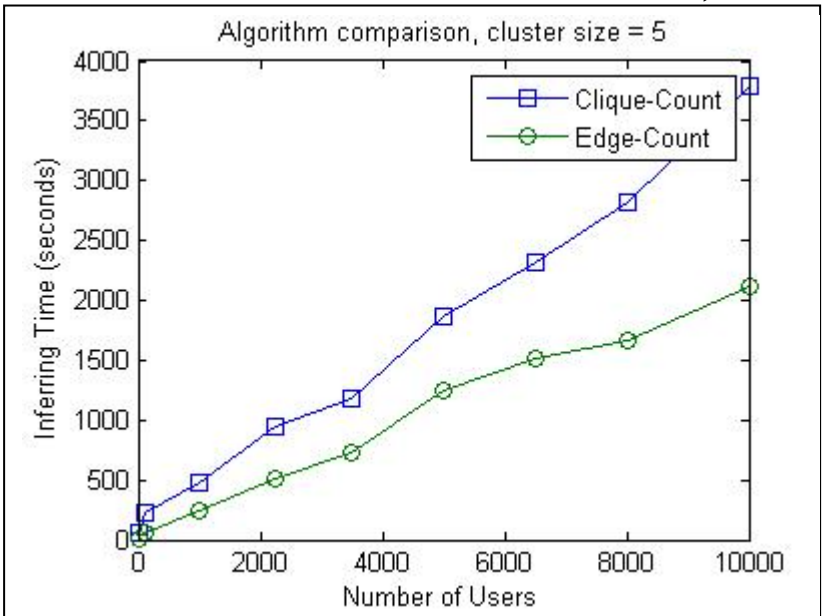


Figure 12-3: A comparison between the Edge counting algorithm and Clique count algorithm with an average cluster size of 5.

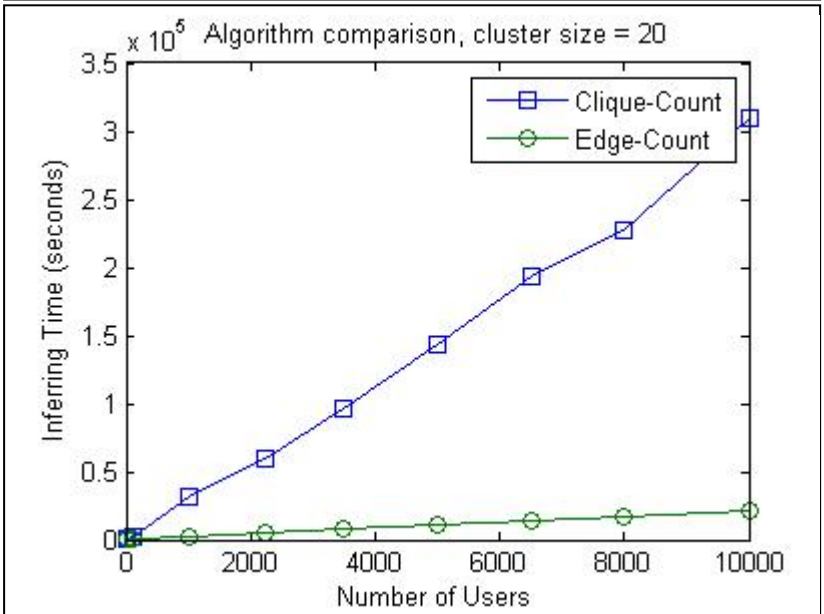


Figure 12-4: A comparison between the Edge counting algorithm and Clique count algorithm with an average cluster size of 20.

12.3.1.2 Graph size increment test case

The next test was to see how increasing of number of vertices in the graph affects the performance. Here the other parameters where fixed to a cluster size of 20 with 10 recommenders and maximum depth of three. The results show that the algorithm is more or less linear until it reaches 95,000 users. As the tests where done on a PC with limited memory the non-linear results with very large numbers of users could be due to lack of memory.

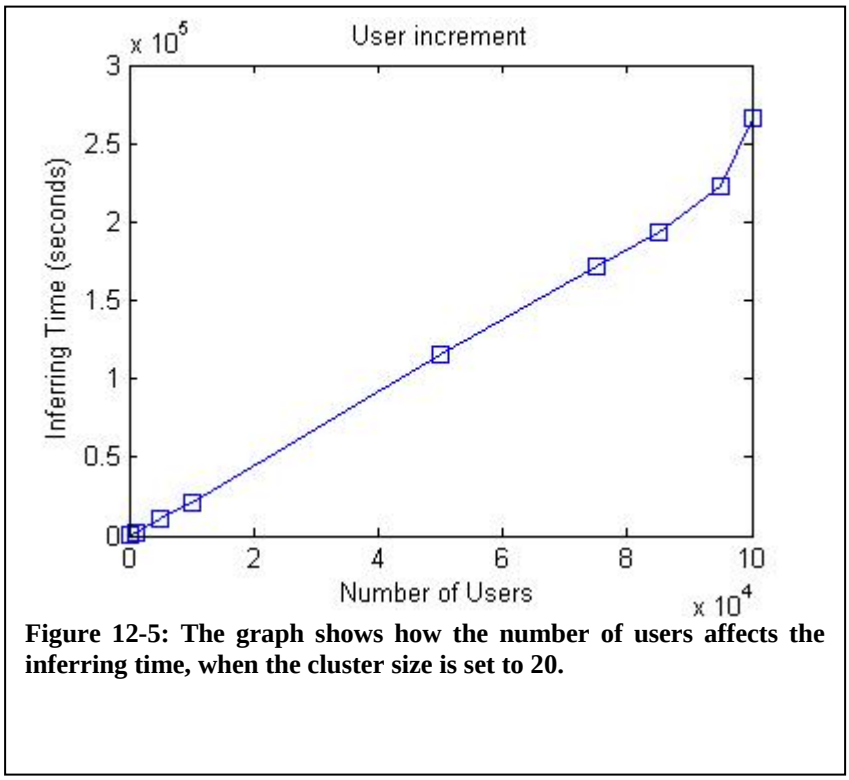


Figure 12-5: The graph shows how the number of users affects the inferring time, when the cluster size is set to 20.

12.3.1.3 Number of nodes and number of edges test case

The figure below shows how the number of edges affects the time when the rest of the parameters are fixed. This shows that even though the number of users is fixed, to 1000, the number of connections between the users will influence the inferring time due to additional computations.

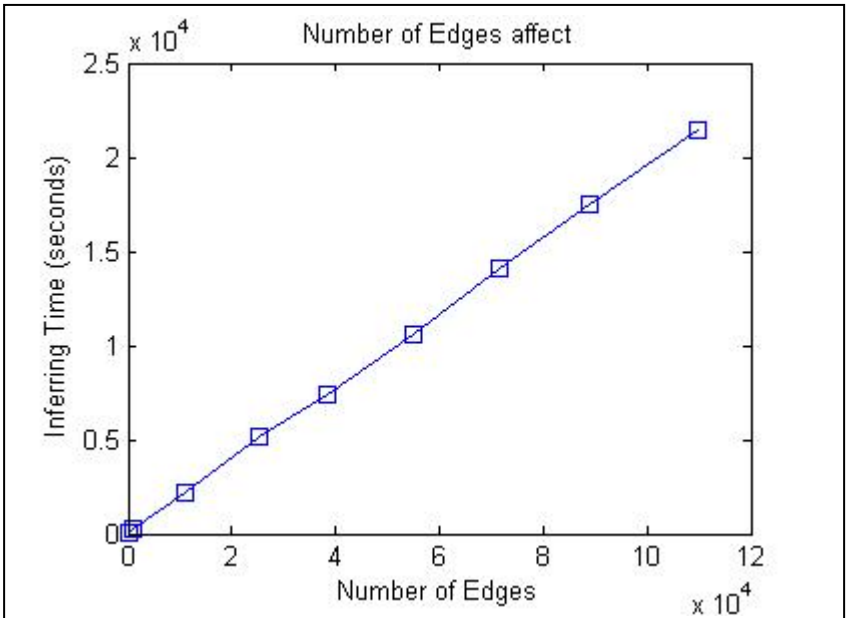


Figure 12-6: The effect of the number of edges on the inference time, when the numbers of vertices are fixed to 10000.

12.3.1.4 Depth influence

The maximum depth will determine how many vertices will be taken into account when calculating the rank. The differences in depth can make a very large difference as the number of users and edges can grow exponentially as the depth increases. Table 12-1 shows that the time required for the computation increased from 33 seconds to 846 seconds when the maximum depth was increased from one to two. Further tests showed that the maximum depth of thee could not be calculated on the current computing platform, as it ran out of heap memory for the application, even when the heap size was set to 800MB.

Table 12-1: Computational performance results of the depth influence.

Number of vertices	100,000	100,000	100,000
Number of edges	1,100,764	1,101,975	1,101,463
Cluster size	20	20	20
Max Depth	1	2	3
Number of recommenders	100,000	100,000	100,000
Inference time (s)	33.078	846.860	Unable to complete as the process ran out of heap memory

As the depth increases the number of users and edges increases. Limiting the depth when the active user's surrounding is very sparse leads to very few recommenders.

When using a specified number_of_recommenders, the depth is dynamically changed depending on if greater depth is needed to calculate rankings. This means that the depth for different active users can be differ depending on whether the surrounding is sparse or not.

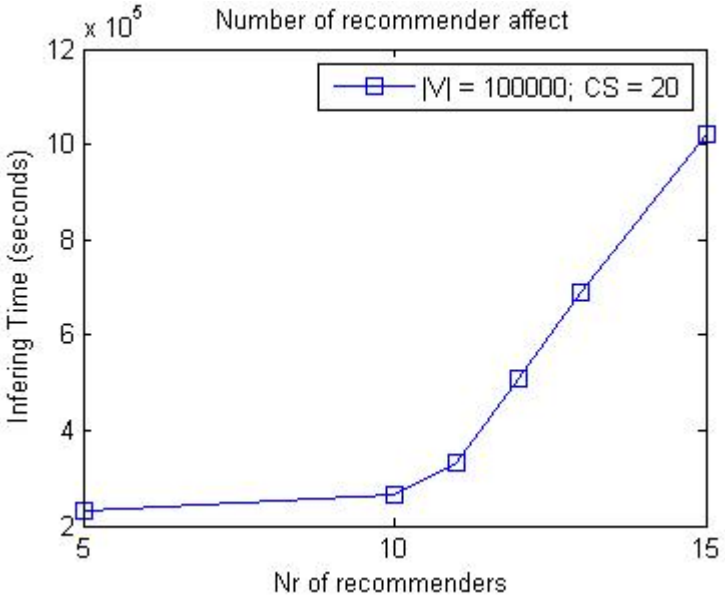


Figure 12-7: The graph shows how the number of recommenders can impact the result of the inferring time. The vertex population is set to 100000 and the cluster size is set to 20.

12.4 Results and conclusion

The results show that the parameters passed to the inference process makes a big difference. Knowing this, it is important to adapt the system to the limitations which exists; as the run-time complexity could be a problem in large scale system. The system could limit the set of recommenders or limit the depth as this affects the number of calculations which need to be performed. It is also important to mention that the tests results might not be truly predictive, as the resources available in the test environment were limited, the input graphs were generated randomly, and other processes were running on this computer. While all of these might affect the results; however, these results should be sufficient to give an approximate solution to our problem.

13 Integrating the result into the recommendation engines

13.1 Using the inferred relations

The recommendation engine should recommend content to the user which has not been seen by this user before (at least as far as the recommendation engines knows). For movies which have not (yet) been seen, a predicted ranking value needs to be inferred. This value will indicate if the content should be recommended to the user. In order to do this properly, different criteria could be taken into consideration. Some information which can be of interests is the inferred relation value and the social network's content rating. A formula which uses the correlation and the ratings for item prediction is the Resnick formula [32]:

$$c(i) = \bar{c} + \frac{\sum_{p \in P(i)} (p(i) - \bar{p}) sim(c, p)}{\sum_{p \in P_i} |sim(c, p)|}$$

Where $c(i)$ is the rating predicted for item i, $\bar{c}$ and $\bar{p}$ are the rated and mean values respectively, $sim(c, p)$ is the similarity between c and p. Resnik considers the average and how much the rating deviates from that average. If the result deviates with a positive value, then this is an indication that the value is high in comparison to the other item that the user has rated.

The similarity in this case could be the Pearson correlation [33]. This is used to ensure that users with a higher correlation will have a stronger influence on the predicted rating than users with lower correlation. If $sim(c, p)$ is replaced with the inferred values in the relation predictions; then higher correlation relations will make the prediction stronger, which is a desired characteristic of the formula.

By replacing the correlation with the relation values we lose information about whether these users have similar tastes for the specified asset type. By combining the correlation with the relation values we can get a function which uses both sources of information. Therefore, we would replace $sim(c, p)$ by a function $f(sim(c, p), rel(c, p))$, where $rel(c, p)$ is the inferred relation between c and p and $sim(c, p)$ is the correlation between c and p. A balance between the weights of the correlation and the relation must exist. Note that if b is set to 0, then only the relation is considered. This is explained by the following formula: $f(sim(c, p), rel(c, p)) = b * sim(c, p) + (b - 1) * rel(c, p)$

13.2 Prediction

As the number of relations in the social network recommendation is limited, there is a high probability that there exist items for which there is only one occurrence in the neighborhood. In that case the Resnik formula could be simplified to the formula below:

$$c(i) = \bar{c} + \frac{(p(i) - \bar{p}) * sim(c, p)}{sim(c, p)} = \bar{c} + p(i) - \bar{p}$$

The predicted rating $c(i)$ of item i will be dependent only on the average of c's ratings, the average of p 's rating, and how p rated the item. Thus the correlation of users is not taken into account (since this case occurred when there was only one occurrence of a rating in the neighborhood). Even though the users correlate very poorly, many of the items which are recommended will have a very high predicted rating since a user with low average ratings among the overall ratings and a high rating on item i will make the result of $c(i)$ high. Based upon this the following formula was utilized:

$$c(i) = \sum_{p \in P(i)} sim(c, p) * p(i)$$

In the above formula the average weighted prediction is directly used without averaging. The drawback is that this formula could give a large value which is not normalized, when the number of neighbors increases. Thus a predicted rating for every item will increase, even when users with low similarity have rated the item. This characteristic is welcomed since many neighbors with small similarity should be able to impact the result, since it is the collective ratings of the users which are interesting. This formula does not have the same problem which the Resnik formula above had; and which the average weighted prediction has, as the similarity will never be omitted.

Once the items' predicted values are calculated, then the x number of items with the highest values are recommended to the user, where x is set by the system.

13.3 Datasets

Two data sets where used to test the prediction: the *Jester* and the *MovieLens* dataset. They had collections of rated movies from users who used their recommendation system. These datasets can be downloaded by anyone who is interested.

The MovieLens dataset[11] contained 1682 movies which were rated by 943 users. The data set was tested on the implemented recommendation engine, as it a well known dataset and provides a lot of data of interest for developers of recommendation engines, such as test sets which allow recommendation engine implementers to compare their accuracy. This set was **not** chosen for the evaluation of the recommendation for the reasons described below.

The content which was chosen for evaluation of the recommendation engine was the Jester jokes dataset. These jokes were obtained from a research group at the University of California at Berkley, who has collected jokes for use with their recommendation engine[12]. As these jokes could be read and rated must faster than watching and rating a movie, this choice of content was expected to reduce the time it takes to complete the evaluation; while also solving other practical problems – such as the difficulty of having enough movies to perform a reasonable test. The test data included 100 jokes and user ratings of those jokes. The ratings will be used to find user correlations between users.

[11] The data sets can be found at GroupLens homepage at http://www.grouplens.org/node/73.
[12] The eigentaste recommendation engine can be found at http://eigentaste.berkeley.edu/user/index.php.

14 Evaluating the recommendations

To evaluate the recommendation engine users must give their opinion of a recommendation which was given. The user therefore reads and rates a set of jokes. These ratings will be added to the existing data set, which was downloaded from the developer of eigenTaste's homepage[13].

The evaluation requires the following:

- Each user must read and rate 10 randomly chosen jokes.
- Each user must choose other users that they trust as recommenders. Note that they can only choose to trust users who are taking part in the evaluation.
- Each user must evaluate the results of the recommendation list.

The evaluation was conducted at Ericsson Research with colleagues who are working in the same personal TV project as the author. This may introduce some bias, but this was neither evaluated nor corrected for in this evaluation. The evaluation was split into two parts. The first part utilized a small set of test users. After analyzing the results of their initial evaluation, then appropriate actions could be taken if the results prove to be bad; by changing the experimental parameters which were used. The second evaluation was conducted with a larger set of users, where the **same** test procedure was followed.

In order to avoid the users needing a Facebook account; a workaround was done to lower the complexity of the evaluation. Thus there was a fixed set of users from whom the test users could select "friends" – all of whom were taking part in the evaluation. The user's section of friends is done by marking a checkbox with the user's name beside it. A checked checkbox indicates that the user has a good taste in jokes. This data was used to create a graph which at a later stage of the evaluation was input to the recommendation system.

The users also need to rate the jokes. To do this, each user is presented with a set of randomly chosen jokes, which they must rate from one to five. Here a higher rating should increase the likelihood that this joke will be recommended to other people (of similar taste). Once the jokes were rated they were manually inserted into the recommendation system. Next the relation inferring was done and rating values predicted for each user. This generated a unique recommendation list for the each user which was given back to the user. Along with the recommendation list some questions were also presented that each user needed to answer. The following questions were asked:

Question1: How good was this recommendation?

Question2: How was this recommendation in comparison to your rated jokes?

Question3: How does this recommendation relate to what you would expect from your recommenders?

[13] http://goldberg.berkeley.edu/jester-data/

The evaluators entered what they thought of the recommendation by entering a numerical value from one to five, in order to compute a mean opinion score (MOS)[14].

14.1 First test

For the first test seven people evaluated the system. They each were given a random selection of 12 jokes each out of 60. The initial set of jokes was narrowed down to 60 from 100, since there should be an overlap of items between the users. Overlaps are something which naturally exists in the datasets and therefore it was important to ensure that such overlap existed in the test dataset.

The mean opinion score gave the following result for the questions:

Q1: 3.8

Q2: 3.8

Q3: 3.7

The conclusion from analyzing these results is that the recommendation system gave recommendations which were rated positively. Thus recommendations resulted in better jokes, than those randomly given to the user, accordingly to question 1. However, the average was below 4. To analyze why this might be so, an average of 4,136,513 ratings from the jester data set was calculated. The result was that the average of rated jokes in the entire dataset was 2.685. This might mean that even though the user did believe that they got better jokes – these jokes where not so much better than the randomly selected jokes, since so many of the jokes where considered bad.

The analysis of the responses to question 2 confirmed that the recommendation was something that their trusted recommenders would recommend. This analysis also gave a result which was below 4. When comparing the recommendations with the jokes which were rated highly by the friends, it appeared that the jokes which were rated high by friends was not necessarily appreciated by the user to whom they were recommended.

The analysis of responses to question 3 gave the lowest result. Although question 1 and 2 covered the third question, the responses to question number 3 revealed that these users thought that the recommendation was good, but not convincing.

The conclusion of this evaluation was that there probably were not enough friends in the set of test subject to generate good overlaps between the items. Larger social network graphs with more overlaps would probably give a more accurate result, since several members from the social network would be likely to have rated an item; thus leading to the start of the second test.

14.2 Second test

In the second test there were 24 users evaluating the system. The full set of 100 jester jokes where used. Each test subject had to rate 6 jokes each; thus providing a good overlap between items.

[14] Mean Opinion Score – The arithmetic mean of numerical indications of perceived quality/correctness.

An analysis of the responses is shown below. These results show that the third question still had a low value in comparison to the others. Just as in the first evaluation, there were people who did not like the recommendation, even though their friends had rated the items highly.

Q1: 3,8

Q2: 3,7

Q3: 3,4

Although there were more people from whom a user could choose when choosing trusted recommenders, the results still showed that the mean opinion score was lower than expected, see the trust network in Figure 14-1.

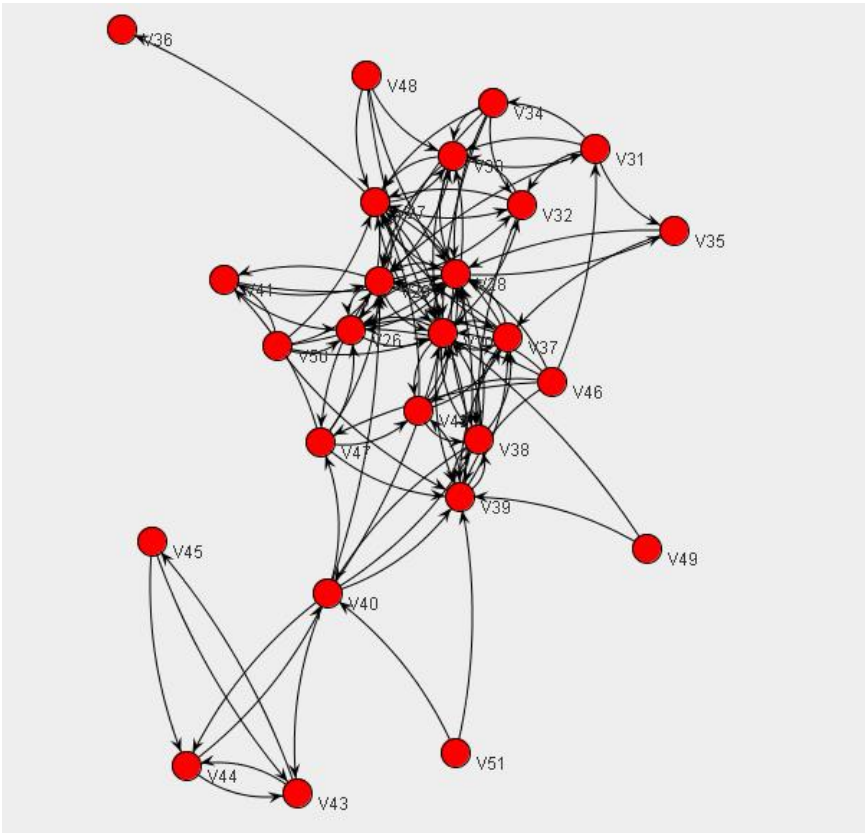


Figure 14-1: The generated graph with the trust relations from the second evaluation.

14.3 Results

The result showed that even though people think that some users have the same taste as they; some people do not agree when with their recommendations. One question which this raises is that users might not choose the correct recommenders. Although this evaluation was conducted with a limited number of test subjects, the average results seemed to all be in the range between 3 and 4.

These results could also been affected by the quality of the jokes, as well as the limited number of jokes which existed in this dataset. What remains to show is if the result would change with a larger data set, as movieLens, as this data set contains many well known and highly appreciated movies.

15 Conclusions

The aim of the thesis was to gain sufficient theoretical knowledge of recommendation engines to create one of my own and integrate it with information obtained from a social network. An analysis of existing social networks which make use of trusts was made in order to get a view of how the problem might be solved.

During the development of the system we examined how performance could be an issue when inferring relations in a social network and how one might use a work around, or approximations, in order to find a reasonable performance solution. Although a full scenario to test performance was not done, as access to the required social network system was not possible (as no social network could be extracted from Facebook; since that would require permission from many users to access their data); therefore, testing was done on a generated social network.

Although the evaluation of the system was limited we believe that there were some interesting results. The most interesting result was due to my assumption that people could choose which persons they trust to give good recommendations. While this was true in many cases, there existed users who did not like the recommendations which where given to them -- although all of these recommendations were based on their friend's ratings of jokes. The result of the limited system evaluation did not reach the level we had hoped for. We would encourage people who are building similar systems to use a much larger dataset, as we believe that this may have been a major factor behind our low results.

We believe that this area will be increasingly important given today's flood of information (which is likely to continue). Covering recommendation systems, data mining/extraction on the web, and the social network analysis within one thesis proved to be a challenge -- since all areas are very rich and there is an extensive body of research in each of the areas. By focusing on only the social network analysis and the recommendation areas we believe more solid knowledge of these areas could have been gained, leading to greater use of advanced collaborative filtering technologies.

16 Future work

This chapter presents some interesting work which did not fit within the time frame or scope of this thesis.

16.1 Improved relationship inferring

Instead of using binary trust values, which only say if the user is trusted, there could be an option to enter a value which says how much the user is trusted. This value would then be used to infer the strength of relations.

In addition; in the networks there always exist users which are called buffers. These users are always popular recommenders for a content type. By looking at how many users trust such a user the system should be able to increase the rank of items selected by these buffers. Note that the resulting buffer ranks are global values which could easily be added to the inference process.

16.2 Combining several Social network sources

Commonly users are member of several communities. LinkedIn might be used for keeping in touch with colleagues and business partners, whereas Facebook might only be for keeping in touch with former classmates or friends. The difference in the purpose of the community might be used to extract friends who form a subset of these communities. Thus the user should be able to choose which communities are interesting to them from a recommendation point of view for a given type of content.

As the recommender should be able to recommend different kinds of content, some types could be more business oriented. Thus while business partners do not necessarily have to be the best movie recommender; the latest news video clips in the business world might be appropriate video content for a business user.

To make it easier to select users as recommenders, the user's entire LinkedIn contact list might be put into the business clips asset type recommendation system by default. Implementing this will further complicate the usability testing and will therefore require greater knowledge about conducting usability tests.

17 Terminology

17.1 Collaborative filtering terminology

Collaborative filtering	The method of filtering information among data sources and user profiles in a collaborative manner.
Memory based	Utilize the entire user-item matrix to generate prediction. Can be both user and item based.
Model based	Develops a model of the ratings to generate predictions. Can be both user and item based.
User based	Uses correlations between users (rows) to make prediction.
Item based	Uses correlations between items (columns) to make prediction.
Neighborhood	The top-N users with the highest correlation to the active user.
Neighbor	A user in the neighborhood.
Training set	Set of rated items which are used to make prediction in a recommender evaluator (approx. 90% of data).
Evaluation set	Set of rated items which the recommender evaluator uses to make prediction on to determine accuracy of prediction (approx. 10% of data).

$$R = \begin{array}{c} \\ u_1 \\ u_2 \\ \vdots \\ u_m \end{array} \begin{array}{cccc} i_1 & i_2 & \dots & i_n \\ R_{1,1} & R_{1,2} & \dots & R_{1,n} \\ R_{2,1} & R_{2,2} & \dots & R_{2,n} \\ \vdots & \vdots & \ddots & \\ R_{m,1} & R_{m,2} & & R_{m,n} \end{array}$$

All the ratings from all the users will be represented by a user-item matrix R. Each user is represented by a row vector and each item is represented by a column vector.

u	Active user
v	Compared user
i	Active item
j	Compared item
$R_{u,i}$	Rating of user *u* on item *i*
$P_{u,i}$	Prediction for the active user *u* on active item *i*, where $i \notin I_u$
$U = \{u_1, u_2 ... u_m\}$	Set of *m* users
$I = \{i_1, i_2 ... i_n\}$	Set of *n* items
U_i	Set of users that have rated item *i*, where $U_i \subseteq U$
I_u	Set of items which user *u* has rated, where $I_u \subseteq I$

I_r	Set of items which user *u* has gotten recommendations, where $I_r \subset I$ and $I_r \cap I_u = \phi$
$U_i \cap U_j$	Set of all the users that have rated on both items *i* and *j*
$I_u \cap I_v$	Set of all the items that have been rated by both users *u* and *v*

17.2 Social Network terminology

Directed Graph	Graph where edges has a direction. The edges are often denoted by an arrow indicating the direction of the edge.
Undirected Graph	Graph where edges has a no direction. The edges are often denoted by a single line connecting the vertices.
Transitive relationship	A relationship where a user does not have a direct relation to a specific use, but through mutual friends.
Clique	A graph problem of determining largest subset where all the users are connected to each other.
Clique division	When the size of the cliques is divided into smaller cliques of size three.

References

[1] Sean M. McNee, John Riedl, and Joseph A. Konstan; "Accurate is not always good: How Accuracy Metrics have hurt Recommender Systems", In CHI '06: CHI '06 extended abstracts on Human factors in computing systems, ACM Press, New York, NY, USA, 2006. Pages 1097-1101.

[2] FOAF-a-matic - Describe yourself in RDF
http://www.ldodds.com/foaf/foaf-a-matic [last accessed 2008-03-26]

[3] Daniel Lemire and Anna Maclachlan; "Slope One Predictors for Online Rating-Based Collaborative Filtering", SIAM, 2005

[4] Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl, "Item-based collaborative filtering recommendation algorithms", Proceedings of the 10th international conference on World Wide Web, Pages 285-295, May 01-05, 2001, Hong Kong, Hong Kong"

[5] Duine homepage
http://www.telin.nl/index.cfm?project=Duine&language=en [last accessed 2008-03-27]

[6] Manos Papagelis and Dimitris Plexousakis: Qualitative Analysis of User-Based and Item-Based Prediction Algorithms for Recommendation Agents. CIA 2004: Pages152-166

[7] XHTML Friends Network project homepage
http://gmpg.org/xfn/ [last accessed 2008-03-27]

[8] FOAF-Project homepage
http://www.foaf-project.org/ [last accessed 2008-03-27]

[9] Google Social Graph project homepage
http://code.google.com/apis/socialgraph/docs/ [last accessed 2008-03-27]

[10] Sesame – openrdf.org - community site to support the development of Sesame. http://www.openrdf.org [last accessed 2008-03-27]

[11] Elmo user guide
http://www.openrdf.org/doc/elmo/1.0-beta2/user-guide/ [last accessed 2008-03-27]

[12] Jena – A Semantic Web Framework for Java
http://jena.sourceforge.net/ [last accessed 2008-03-27]

[13] Stanley Milgram. "The Small World Problem. Psychology Today"; Psychology Today, Volume 1, Number 1, May 1967, Pages 60-67.

[14] Flink homepage
http://flink.semanticweb.org/ [last accessed 2008-03-27]

[15] E. Garcia: "Keywords co-occurrence and Semantic Connectivity: An Introductory Series on Co-Occurrence Theory for Information Retrieval Students and Search

Engine Marketers" 2005-06-05, http://www.miislita.com/semantics/c-index-1.html

[16] JavaScript Object Notation (JSON)
http://www.json.org/ [last accessed 2008-03-27]

[17] Matsuo, Y., Mori, J., Hamasaki, M., Ishida, K., Nishimura, T., Takeda, H., Hasida, K., and Ishizuka, M. 2006. POLYPHONET: an advanced social network extraction system from the web. In *Proceedings of the 15th international Conference on World Wide Web* (Edinburgh, Scotland, May 23 - 26, 2006). WWW '06. ACM, New York, NY, Pages 397-406.

[18] W3C – "SPARQL – Query language for RDF", 2007-06-14
http://www.w3.org/TR/rdf-sparql-query/

[19] "An Introduction to Graph Theory in Complex Systems Studies" http://www.itee.uq.edu.au/~gtheory/GraphTheoryIntroduction.pdf [last accessed 2008-03-27]

[20] Eric W. Weisstein, "Correlation Coefficient."
http://mathworld.wolfram.com/CorrelationCoefficient.html

[21] COFE - COllaborative Filtering Engine
http://eecs.oregonstate.edu/iis/CoFE/ [Last accessed 2008-06-04]

[22] XNF Generator web tool
http://www.accessify.com/tools-and-wizards/developer-tools/xfn/default.php
[Last accessed 2008-06-04]

[23] John Talbot and Dominic Welsh: Complexity and cryptography, An introduction, Cambridge University Press, 2006

[24] Peter Mika, "Social network and the semantic web", Springer, 2007

[25] Junichiro Mori, Yutaka Matsuo, Koichi Hashida, and Mitsuru Ishizuka. "Web Mining Approach for a User-centered Semantic Web"
http://kmi.open.ac.uk/events/usersweb/papers/15_mori_final.pdf [Last accessed 2008-0-12]

[26] Eric W. Weisstein, "Spearman Rank Correlation Coefficient."
http://mathworld.wolfram.com/SpearmanRankCorrelationCoefficient.html [Last accessed 2008-06-04]

[27] Taste - Collaborative Filtering for Java
http://taste.sourceforge.net/ [Last accessed 2008-06-04]

[28] JUNG, Java Universal Network/Graph Framework
http://jung.sourceforge.net/ [Last accessed 2008-06-04]

[29] World Wide Web Consortium, Resource Description Framework (RDF)
http://www.w3.org/RDF/ [Last accessed 2008-06-04]

[30] LinkedIn, social network community
http://www.linkedin.com/ [Last accessed 2008-06-04]

[31] Facebook, social network community
http://www.facebook.com/ [Last accessed 2008-06-04]

[32] Paul Resnick, NeoPhytos, Mitesh Suchak, Peter Bergstom, and John Riedel, "GroupLens: An Open Architecture for Collaborative Filtering of Netnews," Proc. ACM 1994 Conf. Computer Supported Cooperative Work, ACM Press, Pages 175–186 , 1994

[33] John O'Donovan and Barry Smyth. Trust No One: Evaluating Trust-based Filtering for Recommenders. International Joint Conference on Artificial Intelligence IJCAI., Edinburgh, Scotland. 2005.

[34] M. Granovetter. The Strength of Weak Ties. American Journal of Sociology, 78(6), 1973

[35] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering, Pages 43-52, 1998.

[36] J. Golbeck. and Hendler, J. 2006. Inferring binary trust relationships in Web-based social networks. *ACM Trans. Interet Technol.* 6, 4 (Nov. 2006), 497-529.

[37] J. Golbeck. Computing and Applying Trust in Web-based Social Networks. Ph.D. Dissertation, University of Maryland, College Park, 2005.

[38] J. Golbeck, "Combining provenance with trust in social networks for semantic web content filtering," in Int'l Provenance and Annotation Workshop, 2006.

[39] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. “The eigentrust algorithm for reputation management in p2p networks”. Proceedings of the 12th International World Wide Web Conference, May 20-24, 2004.

[40] L. Page, S. Brin, R. Motwani, and T. Winograd. “The pagerank citation ranking: Bringing order to the web”. Technical Report 1998, Stanford University, 1998.

[41] Gredory D. Linden, Jennifer A. Jacobi, and Eric A. Benson,"Collaborative recommendations using item-to-item similarity mappings",Patent applications: EP1121658, US6266649-B1, WO17792-A1, 8 August 2001.

Appendix

First Evaluation Results

Question1: How good was this recommendation?

Question2: How was this recommendation in comparison to your rated jokes?

Question3: How does this recommendation relate to what you would expect from your recommenders?

User	Question1	Question2	Question3
$User_1$	4	4	3
$User_2$	4	5	4
$User_3$	5	4	5
$User_4$	3	3	3
$User_5$	3	3	3
$User_6$	4	4	4
$User_7$	4	4	4
MOS	3.857	3.857	3.714

Second Evaluation Results

User	Question1	Question2	Question3
$User_1$	4	5	5
$User_2$	5	4	3
$User_3$	3	3	3
$User_4$	4	4	4
$User_5$	2	2	2
$User_6$	4	3	4
$User_7$	4	5	4
$User_8$	4	4	4
$User_9$	4	3	4
$User_{10}$	3	2	3
$User_{11}$	4	4	3
$User_{12}$	5	4	3
$User_{13}$	4	4	3
$User_{14}$	5	5	4
$User_{15}$	3	2	3
$User_{16}$	4	4	4
$User_{17}$	3	2	1
$User_{18}$	4	4	3
$User_{19}$	4	3	4
$User_{20}$	2	3	2
$User_{21}$	5	5	4
$User_{22}$	3	4	3
$User_{23}$	5	4	4
$User_{24}$	4	5	4
MOS	3.833	3.667	3.375

COS/CCS 2008-25

www.kth.se